Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Reliability and performance optimization of pipelined real-time systems[☆]

Anne Benoit[a,b,*], Fanny Dufossé[a,b], Alain Girault[c], Yves Robert[a,b,d]

[a] ENS Lyon, LIP Laboratory, France
[b] INRIA, France
[c] INRIA Grenoble Rhône-Alpes, France
[d] University of Tennessee Knoxville, USA

HIGHLIGHTS

- Mapping pipelined real-time systems on distributed platforms.
- Comprehensive set of NP-hardness complexity results.
- Integer linear program for the exact solution of the most general problem instance.
- Two efficient heuristics compared through simulation and with respect to optimal solution.

ARTICLE INFO

Article history:
Received 23 July 2012
Received in revised form
23 January 2013
Accepted 20 February 2013
Available online 5 March 2013

Keywords:
Pipelined real-time systems
Interval mapping
Multi-criteria (reliability, latency, period) optimization
Complexity results

ABSTRACT

We consider pipelined real-time systems that consist of a chain of tasks executing on a distributed platform. The processing of the tasks is pipelined: each processor executes only one interval of consecutive tasks. We are interested in minimizing both the input–output latency and the period of application mapping. For dependability reasons, we are also interested in maximizing the reliability of the system. We therefore assign several processors to each interval of tasks, so as to increase the reliability of the system. Both processors and communication links are unreliable and subject to transient failures. We assume that the arrival of the failures follows a constant parameter Poisson law, and that the failures are statistically independent events. We study several variants of this multiprocessor mapping problem, with several hypotheses on the target platform (homogeneous/heterogeneous speeds and/or failure rates). We provide NP-hardness complexity results, and optimal mapping algorithms for polynomial problem instances. Efficient heuristics are presented to solve the general case, and experimental results are provided.

1. Introduction

A *pipelined real-time system* [22,27] consists of a chain of tasks executing on a distributed platform. Each task is a block of code with a known amount of work to be processed. The role of the first task of the chain is to acquire some data set from the environment (thanks to sensor drivers), to process it, and finally to transmit its result to the second task. Each subsequent task receives its input data from its predecessor task, processes it, and transmits its result to its successor task, except the last task that transmits it to the environment (thanks to actuator drivers). The whole chain of tasks is executed repeatedly, as new data sets enter the system. Each data set is input to the first task and progresses from task to task until its processing is completed.

Executing a real-time system in a pipelined way is essential to increase the throughput, by making the best possible usage of available resources in the distributed execution platform. Tasks are assigned to processors using an *interval mapping*, which groups consecutive tasks of the linear chain and assigns them to the same processor. Interval mappings are more general than one-to-one mappings, which establish a unique correspondence between tasks and processors; they allow communication overheads to be reduced, not to mention the many situations where there are more tasks than processors, and where interval mappings are mandatory. The key performance-oriented metrics to determine the best interval mapping are the *period* and the *latency*. The period is the time interval between the beginning of the execution of two consecutive data sets. Equivalently, the inverse of the period is the *throughput*, which measures the aggregate rate of processing of data. The latency is the time elapsed between the beginning and the end of the execution of a given data set; hence, it measures the response time of the system for processing the data set entirely. Therefore, to minimize the period, we try to create many small intervals so that we can start processing the next data set as soon as

[☆] Part of this work has appeared in ICPP'10.
[*] Correspondence to: LIP, ENS Lyon, 46 alle d'Italie, 69364 Lyon Cedex 07, France.
E-mail addresses: Anne.Benoit@ens-lyon.fr (A. Benoit), Fanny.Dufosse@inria.fr (F. Dufossé), Alain.Girault@inria.fr (A. Girault), Yves.Robert@ens-lyon.fr (Y. Robert).

possible, while for minimizing the latency, we rather try to reduce the sum of communication costs, and hence to split the chain of tasks in the least possible number of intervals. As a consequence, minimizing the latency is *antagonistic* to minimizing the period, and trade-offs should be found between these two criteria.

Each data set has a deadline on the completion time of its execution (the *real-time* constraint). The deadlines are related to the period P and latency L as follows. Data sets periodically enter the system with a given period P. Data set 0 enters the system at time 0 and has a deadline equal to L. Data set K enters the system at time $K \times P$ and has a deadline equal to $K \times P + L$. Accordingly, the deadline of each data set will be met as soon as we derive a schedule whose period does not exceed P, and whose latency does not exceed L. This model is consistent with those applications found in most safety critical real-time systems (e.g., avionics, railway or nuclear applications [8,30]), which enforce a prescribed processing rate and maximum response time. This leads to a global deadline for each application instance (data set), but individual tasks distinct from the output task have no deadlines.

Besides constraints on the performance-oriented criteria, expressed as an upper bound on the period and/or the latency, pipelined real-time systems must also meet crucial *dependability constraints*, which are expressed as a lower bound on the *reliability* of the mapping. Increasing the reliability is achieved by replicating the intervals onto several processors. Augmenting the replication level (defined as the average number of times each interval is replicated) is good for reliability, but bad for period and latency, because fewer processors will be available for executing the task intervals. We thus have three antagonistic criteria: reliability, period, and latency. This antagonism between the criteria makes the problem very challenging.

A typical example of applications with real-time and reliability constraints is encountered in the automotive industry, with the Autosar architecture.[1] Autosar consists of a hardware architecture made of several processors (called ECUs—Electronic Computing Units) connected by a bus, and of several software components, each one being an embedded automotive function. Each function is a pipelined real-time system that starts with some input drivers that will generate a new dataset at each invocation (for instance the wheel angular speed), followed by several software blocks, and terminated by some actuator driver (for instance the hydraulic brake pressure). Each such function must meet a latency (also called the end-to-end timing constraint, from the sensor to the actuator), a period, and a reliability constraint.

We evaluate the reliability of a single task mapped onto a processor according to the classical model of Shatz and Wang [34], where each hardware component (processor or communication link) is *fail-silent* and is characterized by a *constant failure rate per time unit* λ: the reliability of a task of duration d is therefore $e^{-\lambda d}$. For an interval of several tasks mapped onto a single processor, we just have to sum up the task durations, hence obtaining $e^{-\lambda D}$, where D is the sum of task durations in the interval. For a mapping with replication, we compute the reliability by building the *Reliability Block Diagram* (RBD) [29,3] corresponding to this mapping. Here we face the delicate issue that computing the reliability is exponential in the size of the mapping (or equivalently the size of the RBD). To solve this issue, we insert *routing operations* in the mapping to guarantee that the RBD is by construction serial–parallel, therefore allowing us to compute its reliability in linear time. The models are detailed in Section 2 and we discuss related work in Section 3.

Our contribution is multifold. In Section 4, we show how to compute the different objectives (reliability, expected and worst-case latency, expected and worst-case period) for a given multiprocessor mapping. Then, we derive complexity results for homogeneous platforms in Section 5. We prove that:

[1] AUTomotive Open System ARchitecture: http://www.autosar.org.

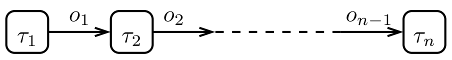
$$\begin{tikzcd} \boxed{\tau_1} \arrow[r, "o_1"] & \boxed{\tau_2} \arrow[r, "o_2"] & \cdots \arrow[r, "o_{n-1}"] & \boxed{\tau_n} \end{tikzcd}$$

Fig. 1. Example of a chain of n tasks.

1. computing a mono-criterion mapping that optimizes the reliability is *polynomial* (Section 5.1);
2. optimizing both the reliability and the period remains *polynomial* (Section 5.2);
3. the problem of optimizing both the reliability and the latency is *NP-complete* (Section 5.3);
4. the problem of assigning processors for a given partition of the task chain into intervals is *polynomial* (Section 5.5).

Moreover, for homogeneous platforms, we provide a linear program to solve the problem of optimization of reliability for given bounds on period and latency in Section 5.4.

For heterogeneous platforms, we prove that the mono-criterion problem of optimizing the reliability is *NP-complete*, and hence all the multi-criteria mapping problems that include the reliability in their criteria are also *NP-complete* (Section 6).

We provide heuristics in Section 7 for the most general problem of optimizing the reliability under constraints on period and latency on a heterogeneous platform, and we conduct experiments on homogeneous and heterogeneous platforms to assess their performance (Section 8). Finally, we state some concluding remarks and future research directions in Section 9.

2. Framework

In this section, we detail the application model, the platform model, the failure model, and the replication model. We end with the formal definition of the mono-criterion and multi-criteria multiprocessor mapping problems.

2.1. Application model

An application is a *chain* of n tasks $\mathcal{C} = (\tau_i)_{1 \le i \le n}$. Each task τ_i is a block of code that (1) receives its input from its predecessor τ_{i-1}, (2) computes a known amount of work, and (3) produces an output data set of a known size. Therefore, each task τ_i is represented by the pair (w_i, o_i), where w_i is the amount of work and o_i is the output data size. By convention, $o_n = 0$ because τ_n emits its result directly to the environment through actuator drivers. Specifying the size of the input data set required by a task is not necessary since, by definition of a chain, it is equal to the size of the output data set of its immediately preceding task. Fig. 1 shows an example of a chain composed of n tasks.

Executing τ_i on a processor of speed s takes w_i/s units of time. Transmitting the result of τ_i on a link of bandwidth b takes o_i/b units of time. Knowing the values w_i and o_i is not a critical assumption since worst-case execution time (WCET) analysis has been applied with success to real-life processors actually used in embedded systems. In particular, it has been applied to the most critical existing embedded system, namely the Airbus A380 avionics software running on the Motorola MPC755 processor [15,35].

2.2. Platform model

The target platform consists of p processors connected by point-to-point communication links. Let $\mathcal{P}$ be the processor set: $\mathcal{P} = (P_u)_{1 \le u \le p}$. We assume that communication links are *homogeneous*: this means that all links have the same bandwidth b. On the contrary, each processor P_u may have a different speed s_u. Such platforms correspond to networks of workstations with plain TCP/IP interconnects or other LANs.