

Communication Scheduling for Time-Triggered Systems

Paul Pop, Petru Eles, Zebo Peng

Dept. of Computer and Information Science,
Linköping University, S-58183 Linköping, Sweden
{paupo, petel, zebpe}@ida.liu.se

1. Introduction

In this paper we concentrate on scheduling for time-triggered systems consisting of multiple programmable processors and ASICs interconnected by a communication channel.

Scheduling for performance estimation and synthesis of embedded systems has been intensively researched in the last years [1]. In our approach, an embedded system is viewed as a set of interacting processes mapped on an architecture consisting of several programmable processors and ASICs interconnected by a communication channel. Process interaction is not only in terms of dataflow but also captures the flow of control, since some processes can be activated depending on conditions computed by previously executed processes. We consider a non-preemptive execution environment in which the activation of processes and communications is triggered at certain points in time, and we generate a schedule table and derive a worst case delay which is guaranteed under any condition. Such a scheduling policy is well suited to a large class of safety-critical applications [4].

The scheduling strategy is based on a realistic communication model and execution environment. We take into consideration the overheads due to communications and to the execution environment and consider, during the scheduling process, the requirements of the communication protocol. Moreover, our scheduling algorithm performs an optimization of parameters defining the communication protocol which is essential for the reduction of the execution delay.

Our system architecture is built on a communication model which is based on the time-triggered protocol (TTP) [5]. TTP is well suited for safety critical distributed real-time control systems and represents one of the emerging standards for several application areas like, for example, automotive electronics [4, 6].

2. Conditional Process Graph

As an abstract model for system representation we use a directed, acyclic polar graph with conditional edges (Figure 1) [3].

Each node in this graph represents a process which is assigned to a processing element. An edge from process P_i to P_j indicates that the output of P_i is the input of P_j . Unlike a simple edge, a conditional edge (depicted with thicker lines in Figure 1) has an associated condition. Transmission of a message on a conditional edge will take place only if the associated condition is satisfied and not, like on simple edges, for each activation of the input process P_i . A process can be activated only after all its inputs have arrived, and issues its outputs when it terminates. Once activated, the process can not be preempted by other processes.

After the termination of a process that produces a condition, the value of the condition is broadcasted from the corresponding processor to all other processors. This broadcast is scheduled as soon as possible on the communication channel, and is considered together with the scheduling of the messages.

3. System Architecture

Hardware architecture

We consider architectures consisting of nodes connected by a broadcast communication channel. Every node consists of a TTP controller, a CPU, a RAM, a ROM and an I/O interface to sensors and actuators. A node can also have an ASIC in order to accelerate parts of its functionality.

Communication between nodes is based on the TTP [5]. TTP was designed for distributed real-time applications that require predictability and reliability (e.g, drive-by-wire). It integrates all the services necessary for fault-tolerant real-time systems. The communication channel is a broadcast channel, so a message sent by a node is received by all the other nodes. The bus access scheme is time-division multiple-access (TDMA). Each node N_i can transmit only during a predetermined time interval, the so called TDMA slot S_i . In such a slot, a node can send several messages packaged in a frame. A sequence of slots corresponding to all the nodes in the

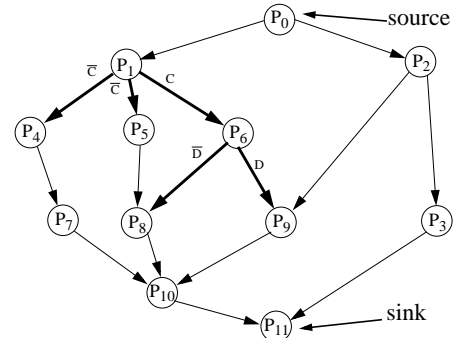


Figure 1. Conditional Process Graph

architecture is called a TDMA round. A node can have only one slot in a TDMA round. Several TDMA rounds can be combined together in a cycle that is repeated periodically. The sequence and length of the slots are the same for all the TDMA rounds. However, the length and contents of the frames may differ.

Every node has a TTP controller that implements the protocol services, and runs independently of the node's CPU. The TDMA access scheme is imposed by a so called message descriptor list (MEDL) that is located in every TTP controller.

Software Architecture

We have designed a software architecture which runs on the CPU in each node, and which has a real-time kernel as its main component. Each kernel has a schedule table that contains all the information needed to take decisions on activation of processes and transmission of messages, based on the values of conditions.

In order to run a predictable hard real-time application the overhead of the kernel and the worst case administrative overhead (WCAO) of every system call has to be determined. We consider a time-triggered system, so all the activity is derived from the progression of time which means that there are no other interrupts except for the timer interrupt.

Several activities, like polling of the I/O or diagnostics, take place directly in the timer interrupt routine. The overhead due to this routine is expressed as the utilization factor U_t . U_t represents a fraction of the CPU power utilized by the timer interrupt routine, and has an influence on the execution times of the processes.

We also have to take into account the overheads for process activation and message passing. For process activation we consider an overhead δ_{PA} , the overhead for sending a message between processes located on the same node is δ_S , the overhead for sending a message to a process on a different node is δ_{KS} , and the overhead of receiving a message from a process on a different node is δ_{KR} .

4. Problem Formulation

As an input we consider a safety-critical application that has several operating modes, and each mode is modeled by a conditional process graph. The architecture of the system is given as described in section 3. The overhead U_t of each kernel and the WCAO of each system call are known. Each process of the process graph is mapped on a CPU or an ASIC of a node.

We are interested to derive a delay on the system execution time for each operating mode, so that this delay is as small as possible, and to synthesize the local schedule tables for each node, as well as the MEDL for the TTP controllers, which guarantee this delay.

The worst case execution delay of a process is estimated taking into account the overhead of the timer interrupt, the WCAO of the process activation, and the WCAO of the message passing mechanism. Therefore, the worst case execution delay of a process P_i will be:

$$T_{P_i} = (\delta_{PA} + t_{P_i} + \theta_{C_1} + \theta_{C_2}) \cdot (1 + U_t)$$

where t_{P_i} is the worst case execution time of the code of process P_i , θ_{C_1} is the overhead for communication from P_i to processes on the same node, and θ_{C_2} is the overhead for communication between

```

schedule_message
slot = slot of the node sending the message
round = current_time / round_length
if current_time - round * round_length > start of slot in round then
    round = next round
end if
while not message fits in the slot of round then
    insert (needed slot length to fit, recommended slot lengths)
    round = next round
end if
put in schedule table (message, round, slot)
end

```

Figure 2. The `schedule_message` function

processes on different nodes:

$$\theta_{C_1} = \sum_{i=1}^{N_{out}^{loc}(P_i)} \delta_{S_i} \quad \theta_{C_2} = \sum_{i=1}^{N_{out}^{rem}(P_i)} \delta_{KS_i} + \sum_{i=1}^{N_{in}^{rem}(P_i)} \delta_{KR_i}$$

In the previous equations, $N_{out}^{loc}(P_i)$ is the number of messages to be sent by the process P_i to other processes on the same node.

$N_{out}^{rem}(P_i)$ is the number of messages transferred to the TTP controller, and $N_{in}^{rem}(P_i)$ is the number of messages transferred from the TTP controller by the kernel, during the execution of process P_i . It has to be noticed that θ_{C_1} refers to the overhead caused by sending the $N_{out}^{loc}(P_i)$ messages generated by process P_i which are directed to other processes on the *same node*. However, θ_{C_2} considers the overhead due to the *remote communications* which not necessarily originate from P_i , but are scheduled to be performed by the kernel during the period P_i is active.

For each message its length b_{mi} is given. Several messages can be packaged together in the data field of a frame. The number of messages that can be packaged depends on the slot length corresponding to the node.

5. The Scheduling Strategy

The problem of conditional process graph scheduling has been addressed by us in [2, 3] without considering a specific communication protocol and execution environment. We do not re-discuss here those algorithms which are part of the function `schedule` mentioned below. That work has been largely extended by considering a realistic communication and execution infrastructure, and by including aspects of the communication protocol in the optimization process. For this reasons, the worst case execution delays T_{P_i} of the processes are computed according to the formula given in section 4.

A major extension concerns the scheduling of the messages on the TTP bus, considering a *given* order of slots in the TDMA round and *given* slot lengths (Figure 2). Function `schedule_message` is called by the function `schedule`, which generates the schedule and corresponding tables based on the given slot order and slot lengths.

In order to get an optimized schedule we have to determine an ordering of the slots and the slot lengths so that the execution delay is as small as possible. We first present two variants of an algorithm based on a greedy approach. A short description of the algorithm is shown in Figure 3. The algorithm starts with the first slot of the TDMA round and tries to find the node which by transmitting in this slot will produce the smallest delay on the system execution time. Once a node was selected to transmit in the first slot, the algorithm continues in the same manner with the next slots.

The selection of a node for a certain slot is done by trying out all the nodes not yet allocated to a slot (the first variant) or the slots recommended by the `schedule_message` function (the second variant). Thus, for a candidate node, the schedule length is calculated considering the TDMA round given so far. Several lengths are considered for a slot bound to a given candidate node.

A second algorithm we have developed is based on a simulated annealing (SA) strategy. The greedy strategy constructs the solution by progressively selecting the best candidate in terms of the

```

greedy
for each slot
    for each node not yet allocated to a slot
        bind (node, slot, minimum possible length for this slot)
        for (1) every slot length or (2) recommended slot lengths
            schedule in the context of current TDMA round
            remember the best schedule for this slot
        end for
    end for
    bind (node, slot and length corresponding to the best schedule)
end for
return solution
end

```

Figure 3. The Greedy Algorithm

schedule length produced by the function `schedule`. Unlike the greedy strategy, SA tries to escape from a local optimum by randomly selecting a new solution from the neighbors of the current solution. The new solution is accepted if it is an improved solution. However, a worse solution can also be accepted with a certain probability that depends on the deterioration of the cost function and on a control parameter called temperature.

6. Experimental Evaluation

For evaluation of our scheduling algorithms we first used conditional process graphs generated for experimental purpose. Table 1 presents the average and maximum the percentage deviations of the schedule lengths produced by Greedy 1 and Greedy 2 from the lengths of the (near)optimal schedules obtained with the SA algorithm, and the average execution time expressed in seconds. The naive designer's approach presented in Table 1 performs scheduling without trying to optimize the access to the communication channel, namely the TDMA round and the slot lengths.

No. of proc.	NaiveDesigner		Greedy 1			Greedy 2		
	aver.	max.	aver.	max.	time.	aver.	max.	time
80	3.16%	21%	0.02%	0.5%	0.25s	1.8%	19.7%	0.04s
160	14.4%	53.4%	2.5%	9.5%	2.07s	4.9%	26.3%	0.28s
240	37.6%	110%	7.4%	24.8%	0.46s	9.3%	31.4%	1.34s
320	51.5%	135%	8.5%	31.9%	34.69s	12.1%	37.1%	4.8s
400	48%	135%	10.5%	32.9%	56.04s	11.8%	31.6%	8.2s

Table 1: Percentage Deviation and Execution Times

Table 1 shows that considering the optimization of the access to the communication channel, the results improve dramatically compared to the naive designer's approach. The greedy heuristic performs very well for all the graph dimensions, and the variant Greedy 1 (that considers all the possible slot lengths) performs slightly better than Greedy 2. However, the execution times are smaller for Greedy 2, than for Greedy 1. The average execution times for the SA algorithm to find the (near)optimal solutions are between 5 minutes for graphs with 80 processes and 275 minutes for 400 processes.

Finally, we considered a real-life example implementing a vehicle cruise controller with one mode of operation having a deadline of 110 ms. The naive designer's approach resulted in a schedule corresponding to a delay of 114 ms, that does not meet the deadline. Both of the greedy approaches produced a delay of 103 ms on the worst case execution time of the system, while the SA approach produced a schedule of 97 ms.

7. Conclusions and Future Work

We have presented an approach to process scheduling for synthesis of safety-critical distributed embedded systems. We have improved the quality of the schedule by taking into consideration the overheads of the real-time kernel and the communication protocol. The scheduling algorithms proposed can be used both for accurate performance estimations and for system synthesis. The algorithms have been evaluated based on experiments using a large number of graphs generated for experimental purpose as well as a real-life example.

In the future we plan to further improve the schedule quality by using a better heuristic for selecting the slots order and lengths, by sharing of messages inside a slot based on the values of conditions and by making use of the knowledge of the TTP parameters during the scheduling process.

8. References

- [1] Balarin, F., Lavagno, L., Murthy, P., Sangiovanni-Vincentelli, A. Scheduling for Embedded Real-Time Systems. IEEE Design & Test of Computers, January-March, 1998
- [2] Doboli, A., Eles, P. Scheduling under Control Dependencies for Heterogeneous Architectures. International Conference on Computer Design, 1998
- [3] Eles, P., Kuchcinski, K., Peng, Z., Doboli, A., Pop, P. Scheduling of Conditional Process Graphs for the Synthesis of Embedded Systems. Proc. Des. Aut. & Test in Europe, 1998.
- [4] Kopetz, H. Real-Time Systems-Design Principles for Distributed Embedded Applications. Kluwer Academic Publ., 1997
- [5] Kopetz, H., Grünsteidl, G. TTP-A Protocol for Fault-Tolerant Real-Time Systems. IEEE Computer, Vol: 27/1, 14-23.
- [6] X-by-Wire Consortium. URL: <http://www.vmars.tuwien.ac.at/projects/xbywire/xbywire.html>