WATERS Industrial Challenge 2017

Arne Hamann, Dakshina Dasari, Simon Kramer, Michael Pressler, Falk Wurst and Dirk Ziegenbein
Corporate Research, Robert Bosch GmbH, Germany
Email: {arne.hamann,dakshina.dasari,simon.kramer2,michael.pressler,falk.wurst,dirk.ziegenbein}@de.bosch.com

I. INTRODUCTION

Automotive embedded applications like the engine management system are composed of multiple functional components that are tightly coupled via numerous communication dependencies and intensive data sharing, while also having real-time requirements. In order to cope with complexity, especially in multi-core settings, various communication semantics are used to ensure data consistency and temporal determinism along functional cause-effect chains. These communication semantics set rules on how and when data is communicated across functions. While "implicit communication" proposed by AUTOSAR targets data consistency, Logical Execution Time (LET) has been proposed to solve the problem of temporal non-determinism by decoupling computation and communication, especially so when the software is deployed across multiple processors. During the design process it is necessary to evaluate the impact of these semantics on the real-time properties of the system.

A. The Challenge

The challenge extends the previous one [1] while mainly focussing on a qualitative and quantitative comparison of the three different semantics: direct, implicit and LET communication, described in Section II. Given an Amalthea meta-model of an engine management system (EMS), with predefined task and label mappings, the solution should

1) propose and demonstrate how implicit and LET communication may be realized, e.g. by adding additional runnables and/or tasks performing copy operations.
2) compute the overheads in terms of extra cycles used for memory access and also in terms of extra memory required due to the proposed implementation.
3) compute end-to-end latencies (age/reaction latency) of the event chains (best, average and worst case). The solution should be able to handle multi-rate effect chains consisting of tasks with harmonic and non-harmonic periods.
4) propose a different label mapping that could possibly reduce the memory access overheads.
5) factor in the effects of contention on the interconnect in the memory access overhead and show the impacts on end-to-end latencies.

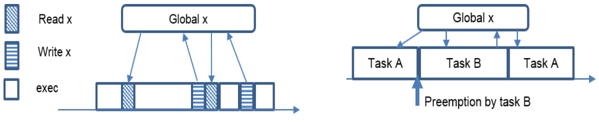
It is very important to clearly state the assumptions made in the implementation and during the overhead calculations. Also solutions that capture the overall system behaviour (under average, best and worst case conditions) using simulation based approaches are encouraged.

B. Model description

We use the AMALTHEA [2] meta-model for describing the engine management system. AMALTHEA provides model elements to express event chains, tasks models, constraints and the hardware platform. The challenge is based on the model of an engine management system provided in the context of the previous industrial challenge [3], [1]. The earlier model is augmented to specify the frequency of label accesses from each runnable. The platform consists of 4 cores, running at 200 MHz, each with a local scratchpad program and data memory and communicate with each other and the global DRAM via a cross-bar interconnection network. The access latencies to local and remote memories are specified in the challenge model. Although the crossbar provides a point-to-point communication channel between each core and memory, there may be contention when multiple cores access any of the memories simultaneously. This contention at the memory ports is resolved using a FIFO arbitration. The application consists of 1250 runnables grouped into 21 tasks/ISRs which communicate via 10000 labels. Constant calibration data, i.e. labels that are only read but never written, is mapped to the global RAM. Variables, i.e. labels that are written by a single task and potentially read by multiple tasks, are mapped to the local memory of the core hosting the writer task. Note that the underlying platform does not support data caching for the data mapped into the global RAM. Additionally all periodic tasks are released synchronously, whereas the angle synchronous task and all ISRs are asynchronously released.

II. BACKGROUND CONCEPTS

A. Explicit or Direct Communication



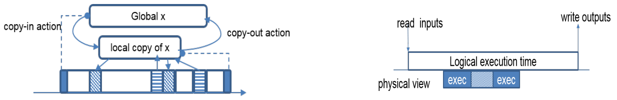
(a) (b)

Fig. 1: a) Direct access: task performs read and writes on a global variable during its execution b) Example showing how task A uses 2 different values at different points in execution

This semantic, often also called direct access, allows unrestricted access to shared variables (labels) across tasks. As seen in Figure 1a, the task works on the global variable of the label. This may work for labels which are strictly read-only, but with labels which may be overwritten, data inconsistency may result. Interleaving of task activations will result in different values of the data. In a single core setting, a simple scenario is one in which a preempting task changes a shared value and so the preempted tasks works on two different values at two different points in time, leading to inconsistencies as seen in Figure 1b.

B. Implicit Communication

This semantic, proposed by AUTOSAR, is primarily focused towards maintaining data consistency to avoid the pitfalls of explicit communication. It essentially follows a read-execute-write paradigm – Implicit communication mandates that the task always makes local copies of the shared data it needs at the beginning of its *execution*, works on the local copies and writes the data back at the end of its execution. This ensures that the task works on the same copy throughout its execution, and also preserves consistent state of the data.



(a) Implicit communication (b) LET communication

Fig. 2: In the implicit case, tasks communicate ate the task boundaries, with LET at activation boundaries

From the access latency perspective, all the variables that are read during task execution will have to be pre-fetched into local memory from its source and then the task may execute by referencing readily available local copies, hence not incurring the cost of remote accesses multiple times. The access latency in case of implicit communication is therefore dependent on multiple factors including, the cost of access to remote and local memory, number of accesses to the label during one execution to the local memory, and the period/activation rate of the task. However on the memory storage front, more local storage is required, since for every task which accesses the label, an extra local copy is required.

C. Logical Execution Time Model

Logical Execution Time (LET) is a real-time programming concept which ensures temporal determinism by decoupling computation and communication. The problem with an unconstrained communication method, i.e, allowing tasks to read and write arbitrarily is non-determinism due to ”execution jitter”. The result is highly dependent on possible interferences of other tasks executing within a tasks activation interval (say from its release to the end of its period). The effects of this jitter becomes more prominent in event chains, leading to large variations in end-to-end delays. With the LET model, tasks always read data at the beginning of the *activation* interval and write data at the end of the activation interval. As with implicit communication, LET requires that a local copy is available for each variable accessed by a task. Using LET, the observable temporal behavior of a task is independent from its physical execution. That is irrespective of the exact time a task executes within its execution interval, the result will be always available only at the end of its execution interval. With LET, the end-to-end latencies in case of synchronous stimuli is always equal the sum of the periods of the tasks involved in the chain. However, with asynchronous stimuli it may happen that each task in the effect chain executes as early as possible in its activation window but the data arrives just after it begins execution (meaning it is operating on an older value of the data). Thus the newer data is consumed only one time period later. The same scenario could occur with every pair of tasks in the chain. Eventually, the worst case latency in the case of such asynchronous arrivals is *twice the sum of the periods of all the tasks in the chain*. LET thus leads to longer latencies in event chains. But on the other hand, with LET, there is no need for complex synchronization mechanisms to handle race conditions or priority inversions, given its well-defined semantics.

D. Intra-task communication

Runnables within a task may communicate with each other in two different ways. With forward communication, the producer runnable completes before the consumer runnable and hence there is no delay in getting the latest data by the consumer and communication is therefore fast. With backward communication however, the consumer runnable executes before the consumer, and thus there is a delay of one period in this case to receive the latest data.

E. Event Chains

An event chain, also called effect chain or signal flow, is an abstraction of a data flow through a system. Typically an EMS has multiple event chains wherein data is sensed by the sensor nodes, passed on to control functions which act on this data and finally the output is used to configure the actuation functions to perform the desired action. Thus these event chains are a sequence of successive stimulus-response segments, where the response of one segment is the stimulus of the next segment. Each of these event chains is associated with an end-to-end latency requirement which is specified via one of the two delay semantics: an *Age* or *Reaction* latency constraint.

1. Reaction Latency Constraint

Reaction latency denotes the time between the occurrence of the response to a specific stimulus. In other words, it denotes the time lapsed between a specific (sensor) input value or signal to a corresponding (actuator) output value, specifying how long a value or signal needs from one end to the other. A reaction latency constraint of k time units to a particular stimulus implies that the response should occur no later than k time units after that stimulus.

2. Age Latency Constraint

For the age latency, the freshness of the data producing the response is important and hence the focus is from the response perspective rather than from the stimulus perspective. In other words, this is the maximum time a specific output (actuator) value is available from a corresponding (sensor) input value or signal. This also equals the validity of a specific value or signal before a new value arrives. A (max) age constraint of ”k” time units for a cause-effect chain mandates that for an occurrence of a response event, the corresponding input data is not older than ”k” time units [4].

REFERENCES

[1] A. Hamann, D. Ziegenbein, S. Kramer, and M. Lukasiewycz, “Demonstration of the fmtv 2016 timing verification challenge,” *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, vol. 00, p. 1, 2016.

[2] AMALTHEA, “An open platform project for embedded multicore systems.” [Online]. Available: http://www.amalthea-project.org

[3] S. Kramer, D. Ziegenbein, and A. Hamann, “Real world automotive benchmarks for free,” in *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, 2015.

[4] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, “Synthesizing job-level dependencies for automotive multi-rate effect chains,” in *Proceedings of the 22th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*.