ABSTRACT

Multicore processors are increasingly important for automotive systems, proof of which is the recent multicore extensions to the AUTomotive Open System ARchitecture (AUTOSAR) standard 4.0. Unfortunately, the current AUTOSAR communication stack specification is still designed to run on a single-core. For that reason, in multicore processors, network applications running on a different core need a proxy to communicate, which incurs substantial overhead due to unnecessary copies.

In this paper, we propose two new approaches for the implementation of the AUTOSAR communication stack in multicore systems: the PDUR server and the MCOM server. Both approaches allocate separate communication buffers for each core—in order to minimize data copies—and their main difference is the inter-core communication layer at which that allocation occurs.

We implemented both approaches, as well as two other existing approaches—the AUTOSAR proxy-based approach and the giant lock approach—on a physical platform; and conducted an extensive evaluation to compare them. The experimental results show advantages and disadvantages in each approach. We expect that the results will be useful to guide the implementation of multicore communication stack.

KEY WORDS

AUTOSAR, Communications, Multicore, Embedded Systems, Real-Time Systems, Automotive

1 Introduction

During the last few years, automotive electronic control systems have increased in sophistication and complexity. Accordingly, the number of electronic control units (ECU) has experimented a substantial growth reaching the amount of around 70 ECUs—typically connected through in-vehicle networks such as the CAN[3] bus—on many modern vehicles. The ever-increasing number of ECUs not only affects the cost of production, but also limits the space inside the vehicle and increases its weight.

For that reason, there is a trend to reduce the number of ECUs by integrating them into a smaller number of units with higher performance. In particular, multicore processors constitute an effective means for realizing such integration due to their lower power consumption. Recognizing this trend, the AUTomotive Open System ARchitecture (AUTOSAR) standard (version 4.0) has recently introduced support for multicore embedded real-time operating systems (OS).

New functions such as the inter-OS application communicator (IOC) have been introduced to extend the single-core OS specification. However, the current AUTOSAR communication stack is still designed to run on a single core that acts as a proxy server for applications running on different cores (i.e., cores without a communication stack) to communicate. Unfortunately, this solution incurs a substantial overhead due to unnecessary copies of data.

The main contributions of this paper are twofold:

- We propose two new approaches for the implementation of the AUTOSAR communication stack in multicore systems: the PDUR server and the MCOM server. The proposed approaches take advantage of the internal structures in the AUTOSAR communication stack to allocate separate communication buffers for each core and share the buffer on cores without a communication stack. This allows minimizing the number of data copies required, and the amount of time spent on critical sections. The main difference between both approaches is the communication layer at which that allocation occurs.

- We implemented both approaches, as well as two other existing approaches—the AUTOSAR proxy-based approach and the giant lock approach[7]—on a physical platform; and conducted an extensive evaluation to compare them.

The results of our evaluation reveal that each approach has advantages and disadvantages; and we expect that the results will be useful to guide the implementation of multicore communication stack.

The remainder of this paper is organized as follows. Section 2 reviews the AUTOSAR communication specification. Section 3 describes the main issues that arise in the communication stack of AUTOSAR multicore systems and existing approaches. In Section 4, we explain the architecture of our two proposed approaches, the PDUR server...
and the MCOM server. Section 5 provides details about the implementation and evaluation of each approach on a physical platform. The experimental results are discussed in Section 6. Section 7 overviews related work, and the paper is concluded in Section 8.

2 AUTOSAR communications

2.1 Overview

AUTOSAR systems are composed of three main layers: the application layer which contains a variety of software components (SWCs); the basic software (BSW) module which provides fundamental services such as OS and communications functionality; and the runtime environment (RTE) which serves as an intermediate layer between the SWCs and the BSW.

Figure 1 depicts the AUTOSAR communications architecture on a dual-core processor. SWCs communicate with each other through messages of configurable size known as signals in AUTOSAR terminology. There are three different types of communication: intra-core, inter-core, and inter-ECU communication. Intra-core communication refers to the communication between SWCs running on the same core, and it is handled within the RTE. Inter-core communication refers to the communication between SWCs running on different cores, and it is handled by the so-called inter-OS application communicator (IOC) within the OS. Inter-ECU communication refers to the communication between SWCs running on different ECUs, and it is handled by the AUTOSAR communications (COM) stack within the BSW module.

In the current AUTOSAR standard, the COM stack is designed to run on one single core (e.g., core 1 in figure 1). For that reason, SWCs running on other cores (e.g., SWC 3 on core 2) need a proxy SWC (e.g., SWC 2 on core 1) to communicate with SWCs in different ECUs. Communications between the client SWCs and the proxy SWC are carried out via the IOC, thus imposing an additional execution overhead due to unnecessary copies of data. One goal of this paper is reducing this overhead.

2.2 The COM stack

The AUTOSAR COM stack is composed of several layers: the COM layer; the protocol data unit router (PDUR) layer; and the network interface and driver layers (CanIf and CanDrv in case of networks based on the CAN protocol). The COM layer packs inter-ECU communication signals from multiple SWCs into interaction protocol data units (I-PDUs). These I-PDUs are stored in the corresponding I-PDU buffers within the COM layer, and transferred to the lower layers (e.g., PDUR, CanIf, CanDrv). The PDUR layer is only used in ECU's that serve as gateways to other networks. Otherwise, the COM layer can directly call the CanIf layer without having to go through the PDUR layer (e.g., the PDUR function PduR_ComTransmit is replaced by the CanIf function CanIf_Transmit). When a signal is received from a SWC that belongs to a different ECU, the same procedure is repeated in reverse order.

2.3 The sending procedure

In this section, we describe the sending procedure of the COM stack with more detail. The sending procedure contains the following two steps:

Step 1. An SWC sends a signal to the COM layer via the RTE. Then, the COM layer updates the I-PDU buffer corresponding to that signal.

Step 2. The COM layer passes a reference to the I-PDU buffer to the lower layers, which finally transmit it through the CAN controller.

The instant at which step 2 occurs can be configured. If step 2 occurs immediately after step 1 completes its execution, the sending procedure is called direct (dSend). In contrast, if step 2 is performed periodically, the sending procedure is called periodic (pSend). These are shown as Figure 2(a). In a dSend transmission, the signal sent by the SWC is packed into its corresponding I-PDU and copied immediately into the CAN controller. In a pSend transmission, the signal sent by the SWC is packed into its corresponding I-PDU, and then once the next transmission period arrives, a function in the COM layer (Com_MainFunctionTx) copies the I-PDU into the CAN controller.

2.4 The receiving procedure

The receiving procedure contains the following two steps:

Step 1. When the CanDrv receives an I-PDU from a different ECU, it is forwarded to the COM layer via CanIf. The COM layer copies the I-PDU into the I-PDU buffer associated to its data identifier (ID).

Step 2. The COM layer unpacks and passes each signal to the application layer.

The instant at which step 2 occurs can be configured. If step 2 occurs immediately after step 1 completes its execution, the receiving procedure is called immediate (iReceive). In contrast, if step 2 is performed periodically, the receiving procedure is called deferred (dReceive). In an iReceive transmission, the COM layer calls a configurable callback function in the SWC after updating the I-PDU buffer. In a dReceive transmission, the step 2 is delayed until the next period is reached and a function in the COM layer (Com_MainFunctionRx) is called.

3 Requirements for a multicore COM stack

AUTOSAR partnership does not consider details of the implementation because the AUTOSAR specification is...
the focus of the specification. Moreover, because the specification of multicore extension considers the reuse of legacy software (i.e., the software designed for single core system) for compatibility, performance is not the focus of multicore extension.

Böhm et al.[7] proposed Big Kernel Lock (BBL) that is a giant lock for AUTOSAR BSW. The proposed system is reported that it is more efficient than the AUTOSAR approach. In the proposed method two cores can enter the COM stack. However, only one core can enter the COM stack at the same time. Therefore, it is long that other cores wait for acquiring the lock.

These methods have both good and bad points as just described. In this paper, we propose two implementations and evaluate these to clarify the characteristics.

### 3.1 Requirements

This paper compares the multicore extensions of AUTOSAR COM stack in term of execution overhead, dependence on the server core and amount of change.

**Execution overhead** An exclusive access control is needed to protect shared data in a multicore system. Additionally, the system will copy the data between local memory and shared buffer in case of the inter-core communications. These processes cause execution overhead. The execution time of these processes varies when the cores perform processing at the same time. For example, while the core 1 is acquiring the lock, the core 2 cannot acquire the lock until core 1 releases the lock. In this paper, we consider only a waiting time for the lock as an increasing factor of the execution overhead.

Because the execution overhead is varied with lock contention, we use the maximum value and the minimum value to describe it. If only one core tries to acquire the lock, there is not waiting time for this core. Reducing the minimum value of the overhead is effective to improve performance. On the other hand, if multiple cores try to acquire a lock at the same time, the core that cannot get the lock will experience the maximum execution overhead, which is equal to the holding time of lock by other cores. It is necessary that the maximum of the overhead is reduced to guarantee real-time behavior.

**Dependence on the server core** For server method, i.e., communication request that is dependent on the server core like the AUTOSAR approach, its response time from start to complete of the communication is also dependent on the state of the server core. For example, when the server core is performing a send processing, a requested process is not perform immediately. For a delayed request process, it is difficult to guarantee real-time behaviour. Moreover, the lower dependence improves parallel processing. Therefore, it is desirable that the proportion of requesting time to the entire communication time.

In giant lock approach, a proportion of requesting process is zero. Cores, however, do not enter the COM stack at the same time. Therefore, the item of dependence can be used for comparison of server methods, whereas it is unsuitable for comparison of server method and giant lock approach.
**Amount of change** The restriction of memory size is severe in a vehicle system. Therefore, it is desirable not to increase the memory size in multicore extension. In addition, it is better that change of lines are few for easy implementation.

### 3.2 AUTOSAR approach

The AUTOSAR approach has to use the inter-core communications via IOC when the cores need to communicate with other ECU. The processing increases the minimum value of the execution overhead. The IOC needs exclusive access control, such as spin-lock, because it uses a shared memory. Section of exclusive exclusion is in only copy processing and it is short. Therefore, the maximum value of execution overhead is not large difference with the minimum value. The dependence of the server core is high because processing of the COM stack is performed by the server core. Legacy code of the COM stack, which is designed for a single core, can be used without change in AUTOSAR approach. Memory usage is small increasing from legacy code.

Figure 3(a) shows architecture of the AUTOSAR approach. In this approach, communication from SWC3 on core 2 requires copying the signal three times. The AUTOSAR approach is good regarding the minimum value of the overhead and amount of change. However, it is bad regarding the maximum value of the overhead and the dependence of the server core.

### 3.3 Giant lock approach

Figure 3(b) shows architecture of the giant lock approach. Process of acquiring the lock is required before entering the COM stack and releasing a lock after leaving the COM stack to implement a giant lock approach. In the giant lock approach, the minimum execution overhead is only processing of acquiring and releasing the lock. These processing is very short, and then the minimum value of the overhead is very small. If a core is holding the lock, other core cannot acquire the lock. The core holds the lock during performing the COM stack and thus the time of waiting for the lock becomes long. Therefore, the maximum value of the overhead becomes large. As shown in Figure 3(b), since the core 2 can access the COM stack directly, giant lock approach is not dependent on the server core to perform inter-ECU communication. Because the giant lock approach can be implemented by adding processing of acquiring and releasing the lock to the existing source code, amounts of change in terms of memory size and source code is small.

The giant lock approach is good regarding the minimum value of the overhead and amount of change. However, it is bad regarding the maximum value of the overhead and the dependence of the server core.

### 4 Proposed multicore extensions of AUTOSAR COM stack

#### 4.1 Methodology

AUTOSAR approach and a giant lock approach are simple implementations. These methods do not take account of the internal structure of the COM stack. We thus propose two alternative implementations by taking advantage of the internal structure of the COM stack.

With respect to the giant lock approach, fine-grained
locks can reduce the lock contention. An implementation method of fine-grained locks, however, strongly depends on the implementation of the COM stack. Therefore, we consider the implementation methods based on AUTOSAR approach.

AUTOSAR approach has a high dependence on the server core because the whole COM stack is run on it. It is necessary that the cores operate a part of the COM stack in order to reduce the dependence. We propose two methods that each core runs a process of the COM. The reason of focusing on the COM module is that the COM module has many processing and requires a long execution time.

4.2 PDUR server method

The COM module has much processing and its source code is complex. In first method, the COM module does not change for easy implementation. The inter-core communication is performed by PDUR module performs inter-core communications and has a buffer for the communications. This method is called the PDUR server method. Figure 3(c) shows architecture of PDUR server method. This method does not change the source code of COM, CanIf and CanDrv because PDUR performs all processing added for multi-core extension.

The communications of the client core needs requesting processing to the server core like AUTOSAR approach. In regard to send processing from the client core, if COM calls PDUR, PDUR performs the inter-core communication and requests remaining processing to the server core. On the server core, it gets the send data from shared buffer. Then, it copies the data and passes it to CanIf. The subsequent processing is the same as the original one. In regard to receive processing to the client core, CAN controller interrupts the server core and CanIf calls PDUR on the core. PDUR performs a inter-core communication processing and requests remaining processing to the client core. PDUR distinguishes a core from ID of receiving data and it performs communication processing if the result is the client core. On the client core, if processing is requested, it gets the receiving data from the buffer. Then, it copies the data and passes it to COM. COM update the associated I-PDU buffer.

Figure 4(a) shows processing of pSend on the core 2 in PDUR server method. If SWC of the core 2 calls COM and passes a signal to COM, COM will update a corresponding I-PDU buffer. Because the I-PDU is pSend, COM performs only updating an I-PDU buffer, and returns to SWC. Then, if Com_MainFunctionTx is called on the core 2, it searches for pSend I-PDU. If I-PDU of pSend is found, the pointer to an I-PDU buffer will be passed to PDUR. After this communication, PDUR copies the data from I-PDU buffer and processing of the core 2 is finish. If the core 1 receives the data, the core copies the data and pass to CanIf.

The PDUR server method can perform processing of COM on each core, without changing the source code of the COM stack except PDUR. Therefore, the dependence on the server core is small compared with AUTOSAR approach. A memory usage increases because of COM on every core. Processing increases, such as the data copy for inter-core communications, distinguishing a core in receive processing. It becomes an execution overhead but it is comparable to AUTOSAR approach.

4.3 MCOM server method

We focus attention on the I-PDU buffer in order to reduce the overhead of data copy. If the core 1 accesses directly the I-PDU buffer on the core 2, it is not necessary to copy a data for inter-core communications. This is called the MCOM server method and its architecture is shown in Figure 3(d).

The communications of the client core, if processing accesses the I-PDU buffer on the client core, it requires a lock before the access. In regard to send processing, after acquiring lock, COM packs signals to an I-PDU buffer and requests remaining process to the server core. On the server core, it gets data from the I-PDU buffer of the client core, and passes it to CanIf. In regard to receive processing, the call of COM is the same as the original one. COM distinguishes a core from ID of receiving data and copies the data to the I-PDU buffer of the client core if the result is the client core. Then, it requests remaining reception processing to the client core in case of iReceive. In case of dReceive, it is not necessary to request a receive processing because the client core performs a receive processing periodically.

The processing of pSend of the core 2 in the MCOM
server system is shown in Figure 4(b). The processing which calls COM of the core 2 and updates an I-PDU buffer is the same as the original. Then, if \texttt{ComMainFunctionTx} is called with the core 1 and I-PDU of pSend is found, the pointer to an I-PDU buffer will be passed to CanIf, and transmission will be required. If Figure 4(a) is compared with Figure 4(b), copying data for inter-core communication is reduced in MCOM server method.

The MCOM server method requires large amount of change of source code due to the modification of COM. On the other hand, memory usage is small compared to PDUR server method. The dependence to a server core is lower than AUTOSAR approach and higher than a PDUR server system. Finally, the execution overhead is reduced because this method does not copy the data for inter-core communication.

5 Implementation and evaluation

5.1 Implementation

We implemented the four methods using a COM stack based on AUTOSAR Release 3.0.2. The COM stack uses CAN controller based on open source hardware IP from OpenCores[2], and the communication is checked with MicroPecker, a CAN bus analyzer.

We implemented IOC service for AUTOSAR approach. Other methods, PDUR server and MCOM server, require inter-core communications for implementation. We also implemented that like IOC.

Processing of \texttt{ComMainFunctionTx/Rx} has an issue in implementation of giant lock approach. The Execution time of this processing depends on the number of I-PDU to check all the I-PDU. Because accessing an
I-PDU buffer is a critical section, original COM stack disables interrupts in accessing the I-PDU buffer. Additionally, the COM stack disables and enables interrupts every time checking an I-PDU without long time of disabling interrupts. Processing of Com_MainFunctionTx/Rx repeats to disable and enable interruption. In addition to the disabling interrupts, it is necessary to acquire a lock to protect the shared data in the multi-core system. The shared data is only buffer of inter-core communications in AUTOSAR approach and PDUR server method. Therefore, these methods do not acquire a lock in execution of Com_MainFunctionTx/Rx. Giant lock approach and MCOM server method have to acquire a lock in execution of that API. These methods disable interrupts and acquire a lock every time accessing I-PDU buffer in order to avoid long holding time of the lock.

5.2 Experimental setup

In order to measure four communication patterns for every core, we define eight signals and eight I-PDUs. One signal corresponds to one I-PDU. A signal is 32 bits and I-PDU is 64 bits. 32 bits of the remainder of I-PDU are filled with 0. The cores are Altera NiosII 50MHz. These cores have 4KB instruction cache and no data cache. The clock of the timer is 50 MHz. Measurement is performed by calling API of COM directly not using RTE.

For every core, executions of dSend, iReceive, pSend, and dReceive are repeated 10,000 times. The time of holding a lock is also measured. While a core communi- cate, another core has nothing of doing in measuring an execution overhead. Its result is minimum value of the overhead. We also measured the duration that a core holds a lock in order to evaluate the maximum of the overhead.

For the amount of change, we examined the memory and code size. The amount of change of the code was measured using cloc[1].

5.3 Experimental results

5.3.1 Execution overhead

Figure 5 shows the execution time of core 1. “Simple”, “glock”, “pdur” and “mcom” denote the original COM stack, the AUTOSAR approach, the giant lock approach, the PDUR server method and the MCOM server method, respectively. Table 1 shows the ratio of execution overhead to the original COM stack. In result of dSend and iReceive on the core 1, the difference is small between each method. In pSend and dReceive, AUTOSAR approach and giant lock approach increased execution time, but the PDUR server decreased. The MCOM server method is increased in pSend and decreased in dReceive. This reason is as follows. Calling Com_MainFunctionTx/Rx is necessary in pSend and dReceive. The execution time of that API is strongly depended on the number of I-PDU as described above. PDUR server method and MCOM server method have I-PDU buffer for every core. If the core 1 calls the Com_MainFunctionTx, the API checks I-PDU buffer of core 1. Therefore, the execution time of Com_MainFunctionTx is short, and then the execution time of overall pSend is decreased by 6.2%. It is the same reason that dReceive is decreased by 6.7% in PDUR server method and decreased by 6.4% in MCOM server method. On the other hand, the execution time is increased by 10.2% in MCOM server because Com_MainFunctionTx needs to check the I-PDU buffer of the both core 1 and core 2.

In the giant lock approach, the execution time of pSend and dReceive is long because of repeated acquisition and release of the lock for every checking the I-PDU buffer. Moreover, the giant lock approach needs the lock in communications with core 1 in contrast to other methods.

Figure 6 shows the execution overhead of core 2. Three methods except giant lock approach have large overhead because of using inter-core communications. In dSend and iReceive, giant lock approach is the best. The overhead of AUTOSAR approach and PDUR server method are more than 40% because of data copying. In MCOM server method, the overhead is reduced because of no copying data for inter-core communications. In pSend and dReceive, the overhead of PDUR server method is reduced. In MCOM server method, the overhead of pSend is large, but dReceive is small. This reason is same as the case of core 1.

Table 2 shows the worst case each core holds the lock. In the giant lock approach, it is large because it runs the COM stack holding the lock. In the AUTOSAR approach and the PDUR server method, the value is small because holding a lock only happens when accessing shared buffer. In the MCOM server method, it is necessary that the process calls CanIf and CanDrv with locking an I-PDU buffer for passing the data. Therefore, the time is large in the MCOM server method.

5.3.2 Dependence on the server core

Table 3 shows the rate of core 1 process in communications with core 2. In giant lock approach, it is 0% because the core 2 execute all process. The dependence has fallen in
order of AUTOSAR approach, MCOM server, and PDUR server. The result of sending and receiving differs in PDUR server and MCOM server because the rate of the execution time of COM is different between sending and receiving.

5.3.3 Amount of change

Table 4 shows the memory usage. The IOC code is added in AUTOSAR approach, and code of acquiring and releasing the lock is added in giant lock approach. The memory usage of these processing is small. On the other hand, memory usage increases 38.4% in the PDUR server method because it has the COM module for every core. The MCOM server also has the COM module for every core, but it is implemented using code shared. Therefore memory usage increases 10.5%.

The amount of change of a source code is shown in Table 5. While AUTOSAR approach and giant lock approach have few changes, PDUR server and MCOM server need more changes. The MCOM server method needs more changes than PDUR server method due to the modification of COM module.
6 Discussion

The execution overhead is the most important issue. The evaluation clarified the characteristics of each method.

In AUTOSAR approach, the overhead of communication with the core 1 is zero. The maximum value of the overhead is small, but the overhead of communication with the core 2 is large. Additionally, the dependence of the core 1 is high. It is not suitable that SWC which frequently communicates with other ECU is assigned on the core 2.

The giant lock approach has lower execution overhead on the whole. But execution time of pSend and dReceive is dependent on the number of I-PDU. Number of I-PDU tends to be big because I-PDU buffer is not divided by the cores. Additionally, if one core is running the COM stack, the other core cannot run the COM stack and wastes time in waiting. Therefore, the execution time tends to be longer. The giant lock approach is effective when communication processing is almost event-driven and lock contention is rare.

In PDUR server method, it is not necessary to access the I-PDU buffer of the other core. The execution times of pSend and dReceive is shorter than giant lock approach (except for dReceive on core 2). This difference is larger if number of I-PDU increases. For this reason, PDUR server method is effective when communication processing is almost periodic.

In MCOM server method, the execution time of pSend on core 2 is the worst but average of execution time is the best. The execution time of pSend is worse if number of I-PDU increases. Average is good because of no copying data in this method. The time of holding the lock is long on core 1.

7 Related works

About multicore extension of OS, many existing methods were proposed based on open source codes, such as Linux and BSD. These OSes used Big Kernel Lock (BKL) in first implementation for supporting multicore system. The BKL is a giant lock for exclusive access control of the kernel, which is also the reason of performance degradation. Multicore extension of network protocol is also proposed. Interrupt service routine is not scaling for multicore system before Linux kernel 2.6.35. This scaling problem is solved with Receive Packet Steering (RPS)[5] and Receive Flow Steering (RFS)[6] techniques. Application is also extended. L7-filter[4], which is packet level filtering, was extended efficient parallelized for multicore system. Core assign techniques are reported in [12], [11]. These techniques improve cache hit rate by assigning similar process to same core. AUTOSAR partnership opens the specification but does not open the implementation. Therefore, it is better that multicore extension method does not depend on the implementation.

The AUTOSAR system is extended for supporting multicore system[8, 9]. Böhm et al.[7] proposed Big Kernel Lock (BBL) that is a giant lock for AUTOSAR BSW.

8 Conclusion

We proposed two methods for the implementation of the AUTOSAR communication stack in multicore systems. We are planning to publish program we used in this paper as open source by TOPPERS project[10]. Experimental
results reveal the pros and cons of each method.

In existing method using giant lock, the execution time of periodic processing is strongly dependent on the number of I-PDUs. Since the electronic control system of vehicles becomes more sophisticated, the number of I-PDU will tend to increase. Furthermore, periodic processing dominates vehicle system’s software. Although giant lock approach can improve performance, it has those disadvantages. In contrast, in proposed method, the PDUR server method, the execution time of periodic processing is shorter than giant lock approach. However, the execution time of event-driven processing on client core is longer than giant lock approach. Another proposed method, MCOM server method, seems to be a good trade-off between the giant lock and the PDUR server method.

References


