An Efficient Scheduling of AUTOSAR Runnables to Minimize Communication Cost in Multi-core Systems

Hamid Reza Faragardi, Björn Lisper
MRTC/Mälardalen University
P.O. Box 883, SE-721 23 Västerås, Sweden
Email: {hamid.faragardi, bjorn.lisper}@mdh.se

Kristian Sandström, Thomas Nolte
ABB Corporate Research,
SE-721 78 Västerås, Sweden
Email: kristian.sandstrom@se.abb.com, thomas.nolte@mdh.se

***Abstract*—The AUTOSAR consortium has developed as the worldwide standard for automotive embedded software systems. From a processor perspective, AUTOSAR was originally developed for single-core processor platforms. Recent trends have raised the desire for using multi-core processors to run AUTOSAR software. However, there are several challenges in reaching a highly efficient and predictable design of AUTOSAR-based embedded software on multi-core processors. In this paper a solution framework comprising both the mapping of runnables onto a set of tasks and the scheduling of the generated task set on a multi-core processor is suggested. The goal of the work presented in this paper is to minimize the overall inter-runnable communication cost besides meeting all corresponding timing and precedence constraints. The proposed solution framework is evaluated and compared with an exhaustive method to demonstrate the convergence to an optimal solution. Since the exhaustive method is not applicable for large size instances of the problem, the proposed framework is also compared with a well-known meta-heuristic algorithm to substantiate the capability of the frameworks to scale up. The experimental results clearly demonstrate high efficiency of the solution in terms of both communication cost and average processor utilization.**

***Keywords*-AUTOSAR; runnable; mapping; multi-core scheduling; feedback-based search; Simulated Annealing; SMSA.**

I. Introduction

In the area of automotive systems, AUTOSAR [1] is becoming the standard software architecture specifying and dictating how embedded software is developed. As the automotive industry now is facing a migration from traditional single-core processors to parallel multi-core processors, some extensions to how AUTOSAR should be used in the new context are required in order to gain the full potential advantages of multi-core processors. The multi-core technology raises new challenges related to both timing predictability as well as to the possibility of obtaining a reasonable performance when executing embedded software. Such a challenge is inherent in the communication between software components located on different cores of the multi-core processor.

AUTOSAR software contains a set of software components that each component is constructed by hosting a set of runnables - each runnable being a small piece of executable code. Hence, a particular software constructed according to AUTOSAR can be considered as a set of runnables. The runnables should be mapped into a task set. We call the process of assigning runnables to tasks, mapping. The mapping directly effects on the schedulability of the crated task set. The generated task set should then be allocated to the cores of a multi-core processor. A balanced allocation of the task set onto a multi-core (i.e., load balancing) usually results in an acceptable performance in terms of makespan however, such a solution may increase the overall communication cost inherent in communication in-between the runnables when the runnables (and therefore, also tasks) are able to communicate with each other. Increasing the communication cost does not only aggravate the overall performance but it can also reduce schedulability of the system.

The overall goal of the work presented in this paper is to provide a solution to map AUTOSAR runnables onto tasks that are then allocating on the cores of a multi-core processor. In doing this, we intend to make a schedulable solution which is able to minimize the inter-runnable communication cost. We distinguish between different communication cost depending on if the runnables that are communicating with each other are allocated within the same task, or if they are allocated in different tasks on the same core or on different tasks on different cores.

To address the challenges related to the overall goals of this paper, a comprehensive framework is required including solutions for both mapping of runnables to tasks and allocating of tasks onto the cores. It is worth noting that mapping and scheduling must be contemplated together because if the task set is not properly formed, even an optimal task scheduling in terms of communication cost may not lead to an acceptable solution, and vice versa. In this paper, we propose a solution framework to minimize the overall communication cost while at the same time satisfying all timing and precedence constraints. The allocating is carried out based on a new evolutionary algorithm inspired from Simulated Annealing (SA). Moreover, to guide the search towards an optimal solution, the framework has integrated a refinement function which allows us to merge the initial task set. Such a design leads to a feedback search which evaluates each individual of search space based on the feedback generated by the refinement function, hence the probability of achieving the general optimum substantially increases.

The main contributions of this paper can be expressed as follows:

1) We propose a feedback-based solution framework for execution of embedded software on a multi-core pro-

978-1-4799-5359-2/14/$31.00

cessor, subject to minimizing communication cost. Our solutions cover both mapping and scheduling while the most of the existing solutions found in the related work only focus on scheduling.

2) We present a discussion on some alternative approaches which can be applied to tackle with the problem, whether they are feasible, and what would be the expected performance for them, indicating directions for future work.

The rest of this paper is organized as follows: In Section II a brief survey on related work is presented. The problem is described in detail and assumptions are defined in Section III. The solution framework to form the task set and the scheduling is introduced in Section IV. In Section V the performance of the proposed algorithm is assessed in comparison with other alternative approaches. Finally, the concluding remarks and future work is discussed in Section VI.

II. Related Work

A large number of studies have been conducted to solve the challenges related to static allocation of real-time tasks to a set of processors [2], [3]. However, only few of them have taken precedence constraint into account, for example [4], [5], [6]. In these papers, the task set is often described by an acyclic directed graph where the tasks indicate nodes, and the edges between the tasks display causal dependencies. Yosemi and Sorel in 2008 presented an algorithm for schedulability analysis of hard real-time tasks in the presence of precedence constraints on a single processor [4]. They also considered the context switch overhead and, according to fixed priority, the number of context switches was accounted. In [5] two algorithms based on the Branch and Bound (BB) technique were proposed for the static allocation of communicating periodic tasks in distributed real-time systems. The first technique assigns tasks to the processors and the latter schedules assigned tasks on each processor. Due to the exponential nature of BB this approach is only feasible for small and moderate size of problems however, it fails in finding a solution for most real-world sized systems.

The mapping methodology has been widely applied to synthesis and hardware/software co-design of heterogeneous MPSoC systems [7]. It has also been employed to software synthesis of embedded distributed real-time systems [8]. The approaches introduced for conventional real-time systems cannot be directly applied for AUTOSAR systems because, in addition to task assignment, there is another level in AUTOSAR systems which is the mapping of runnables into a task set. On the other hand, an improper task set even with an optimal task assignment may not result in a reasonable performance for the system. Therefore, it is desirable to contemplate both mapping of runnables into a task set and assignment of the task set onto a multi-core phases together.

In the context of AUTOSAR, several studies for mapping of runnables exist, however, they target distributed single-core processing nodes [9], [10], [11], [12], [13]. Among them [13] is more related to our work because, it also considers the phase of runnable mapping to the OS tasks. However, its goal is to minimize the end-to-end latency in a distributed automotive systems. The authors apply a genetic algorithm to cope with the problem. To the best of our knowledge, the mapping of AUTOSAR runnables onto multi-core processors with the goal to minimize the overall communication cost has not yet been carefully studied. This is the focus of this paper.

III. Problem Modeling

Let's suppose a set of software components, each of which comprises a set of runnables. Therefore, the problem to be solved can be considered as a set of strictly periodic runnables which could be concurrently run on different cores. Let $R = \{R_i : i = 1,2,...,m\}$ be the set of $m \geq 2$ runnables to be allocated among a set of $N \geq 2$ processing nodes, $\rho = \{\rho_j : j = 1,2,...,N\}$ of a multicore chip. Let's suppose that we have a homogenous multi-core system in which all the cores have the same processing power. The runnable R_i has a Worst Case Execution Time (WCET) denoted by e_i. In addition, a set of independent transactions $\{\Gamma_i : i = 1,2,...,M\}$ represent end-to-end latency requirements between a sequence of runnables. In fact, each transaction is a directed acyclic graph in which each node is a runnable and links show data dependency between them. This dependency does not imply triggering, in the sense that a successor can start irrespective of the state of its predecessor however. To fulfill the mission of a transaction all successors must start to run with the fresh data generated by its predecessor. The communication between the runnables can be performed based on non-blocking read/write semantics [14]. Fig. 1 shows an illustrative example. Without loss of generality we can assume that all runnables are covered by at least one transaction, because if a runnable is not included in any transaction, then we assume a new transaction which contains this runnable.

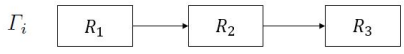

Fig. 1. A sample of a transaction.

The transaction Γ_i has a relative end-to-end deadline denoted by D_i before which the execution of the transaction must finish, i.e., all runnables of the transaction must finish their execution before this deadline. The transaction deadline is corresponding to the mission of that transaction. For example, the mission could be the braking system in a car where the whole process should be done before a specific end-to-end deadline. We assume that the transactions are periodic in nature however, the transaction period (denoted by P_i) is not given as an input of the problem. Nevertheless, a conservative assumption is that the period of the transaction is equal to its relative deadline, i.e., $P_i = D_i$.

Another challenge that should be covered by our model is the dependency between the transactions which means that the transactions may share the same runnable(s). In such cases, the problem can be categorized into two main groups. The first group is when there is not any internal state within the shared runnable(s) whereas, in the second group we have a set of internal states within the shared runnables. Dealing with the first group is much easier than the second group as we

can make a copy from the shared runnable(s) and thereby, make independent transactions. Although this approach might generate extra overhead, it significantly decreases the problem complexity. Fig. 2 shows an illustrative example. To tackle with the second group we need to preserve a kind of mutual exclusion for different runs of the shared runnable(s). Hence, the shared runnables can be considered as a sort of critical instance of a transaction. In other words, we expect if the transaction Γ_i and Γ_j have a shared runnable R_k then when Γ_i is running R_k, Γ_j can not run the R_k until the execution of R_k is finished within Γ_i. The second group is out of the scope of this paper and we only cover the first group here.

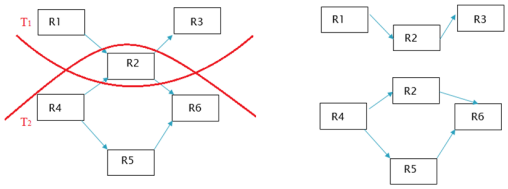

Fig. 2. Two transactions with a common runnable

To complete the problem representation, another graph is also required to represent data communication between the runnables. In this paper we assume that a pair of runnables may transfer data with each other, while there is not any precedence among their execution order. An undirected graph called Runnables Interaction Graph (RIG) shows and represents this information. Each node of the RIG represents a runnable and the arcs between the runnables show data communication between them. In other words, there is not necessarily data dependency between them. It is worth noting that the direction of transferring data between a pair of runnables does not matter in this graph, because it is only used to minimize the overall communication cost between the runnables. Furthermore, there is a label on each arc called communication rate - cr_{ij}, that indicates the rate of data that should be transferred between the runnables per *hyper-period*. The hyper-period is the Least Common Multiple (LCM) of the periods of all the transactions which is denoted by h. Considering data transfer rate per hyper-period allows us to compare communication cost between various runnables irrespective of their periods. Fig. 3 illustrates a RIG instance.

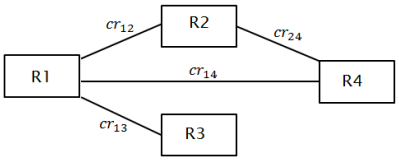

Fig. 3. A sample of a RIG in an AUTOSAR system.

The communication cost to transfer data between each pair of runnables depends both on the mapping of runnables onto the cores and the final task set structure. It can be modeled as follows:

- If the runnables are hosted in the same task, then we consider cost α for transferring each unit of data.
- If the runnables are hosted in different tasks on the same core, then the cost of communication is β.
- If the runnables are allocated to different cores, then γ is the cost of communication.

It should be mentioned that since we would like to address the problem in a general manner without restricting the problem to a specific hardware or a specific operating system, we simply assume the existence of such constant transferring costs (α, β, and γ). However, in practice they strongly depend on the memory management mechanism, the cache mechanism and the hardware configuration.

IV. Solution Framework

In this section two solution frameworks are introduced, each of which comprises both the mapping and scheduling phases. In the following two subsections they are explained in detail.

A. Framework 1: SMSA

In the first framework, the mapping function is simply carried out by considering each transaction as one task. In this case, the task deadline is set equal to the transaction deadline, and the task period denoted by T_i is equal to the transaction period (as is mentioned before the transaction period is considered equal to the transaction deadline). In this way, if a task meets its deadline then the corresponding transaction meets its end-to-end deadline as well. The next step in order to complete the framework is the task scheduling phase. To fulfill the scheduling phase, we use a new evolutionary algorithm called Systematic Memory Based Simulated Annealing (SMSA). Based on recent studies, SMSA outperforms pure Simulated Annealing (SA) [15] [16] when it is applied to solve scheduling problems. SA is a stochastic memory-less algorithm. In that sense, SA does not use any information gathered during the search. SA starts from an initial solution, and it proceeds in several iterations. A random neighbor is generated at each iteration (SA visits only one of the neighbors). The moves that enhance the cost function are always passed. Otherwise, the neighbor is selected with a probability that depends on the current temperature and the amount of reduction of the objective function.

SMSA covers some major drawbacks of SA in order to gain more efficiency. Firstly, SMSA visits the subset of the neighborhood and selects the best member of this subset (systematic nature) versus SA that only visits one of the neighbors. This subset is selected randomly and thus the stochastic nature of SA is still exploited. Secondly, SA sometimes falls into loops and accepts a specific solution more than once; it is the main disadvantage of SA. To avoid the cycling all the visited solutions during the search process must be stored however, it requires a large amount of memory and on the other hand searching in such a large memory takes a lot of time. The wide range of experimental results regarding the performance of SA manifested that the probability of re-visiting recently visited solutions is dramatically higher than the old ones. Accordingly, SMSA utilizes a limited memory to keep recently visited solutions to prevent cycling (memory-based).

SMSA is used as a scheduling algorithm with the following configuration:

- *Problem Space:* The set of all possible allocations for a given set of tasks and processing cores is called the problem space.

- *Solution Representation:* Each point in the problem space is corresponding to an assignment of tasks to the cores that potentially could be a solution for the problem. The solution representation strongly affect the algorithm performance. We represent each solution with a vector of M elements, and each element is an integer value between one and N. The vector is called SR. Fig. 4 shows an illustrative example for a solution. The third element of this example is two, which means that the third task (corresponding to the third transaction) is assigned to the second core. Furthermore, this representation causes satisfaction of the *no redundancy constraint* in the sense that each task should be assigned to no more than one core.

T_1	T_2	T_3		T_M
1	1	2	…	1

Fig. 4. Representation for assigning the tasks onto the cores.

- *Initial Solution:* It is generated randomly. However, in the future work a more effective heuristic algorithm is used to generate the initial solution.
- *Neighborhood Structure:* The neighbors of the current solution are a sub set of the problem space that are reachable by moving any single task to any other processing core. Therefore, each solution has $M(N-1)$ different neighbors, because each task can run on one of the other $N-1$ cores.
- *Selecting Neighbor:* SMSA in each step, instead of considering all neighbors (i.e., $M(N-1)$ neighbors), selects one task randomly and then it examines all neighbors of the current solution in which the selected task is assigned to another core. Hence, it visits $N-1$ neighbors, and then the best solution of this subset is designated irrespective of whether it is better than the current solution. We call this process *stochastic-systematic selection*, because we use a combination of systematic and stochastic process to select the neighbor.
- *Total Cost Function:* The total cost is a function that returns a real value for the assignment SR_z, and this value is used to evaluate each solution. The total cost function can be computed by Eq. 1.

$$TC(SR_z) = \sum_{i=1}^{m} \sum_{j=i+1}^{m} \frac{T_{Y_i(SR_z)}}{h} \times C_{R_{ij}}(SR_z) + \sigma \times P(SR_z) \tag{1}$$

where $Y_i(SR_z)$ is a function that returns the index of the task to which R_i is assigned by the assignment of SR_z, and $P(SR_z)$ is the penalty function that reflects the amount of violation from the acceptable solution. It means if the value of the penalty function is zero, then the assignment SR_z meets all the end-to-end deadlines. Otherwise, some of the deadlines are missed. σ is the penalty coefficient used to guide the search towards valid solutions. This coefficient tunes the weight of the penalty function with regards to both range of cost function and the importance of the constraint violation. For example, in a soft real-time system, where missing a low number of deadlines may be tolerable, the coefficient should be set to a lower value. Furthermore, $C_{R_{ij}}(SR_z)$ denotes the cost of transferring data which is defined by Eq. 2.

$$C_{R_{ij}}(SR_z) = \begin{cases} \alpha \times cr_{ij} & \text{if } I \\ \beta \times cr_{ij} & \text{else if } II \\ \gamma \times cr_{ij} & \text{else} \end{cases} \tag{2}$$

where I denotes a condition in which R_i and R_j belong to the same task whereas, II denotes a condition in which the corresponding tasks of R_i and R_j are located on the same core. Let's suppose that Earliest Deadline First (EDF) scheduler is used as a local scheduler on each core. Based on the EDF utilization test, we can make sure that if the utilization of a processing node derived by the proposed solution is less than one, then the solution meets all deadlines. It should be mentioned that although this schedulability test is merely valid for an independent task set, as we map a whole transaction to one task, the precedence relations between the runnables of a transaction do not generate precedence constraints between the tasks, and thus the generated task set is independent. Therefore, Eq. 3 is applied to calculate the penalty function.

$$\begin{aligned} P(SR_z) &= \sum_{i=1}^{N} max\{0, U_{\rho_i}(SR_z) - 1\} \\ U_{\rho_i}(SR_z) &= \sum_{\forall k, \tau_k allocated to the \rho_i} \frac{E_k(SR_z)}{T_k} \end{aligned} \tag{3}$$

where $U_{\rho_i}(SR_z)$ indicates utilization of the ith processing core, and $E_k(SR_z)$ denotes the worst case execution time of the kth task for the assignment SR_z, that is calculated by

$$\begin{aligned} E_k(SR_z) &= t_{comput} + t_{commun} \\ t_{comput} &= \sum_{\forall l, R_l \in \tau_k} e_l \\ t_{commun} &= f(\frac{T_k}{h} \times \sum_{\forall i, R_i \in \tau_k} \sum_{\forall j, R_j \in R} C_{R_{ij}}(SR_z)) \end{aligned} \tag{4}$$

where t_{comput} implies the computation time which is independent from assignment of tasks to the cores thus, can be calculated in advance, and t_{commun} represents the communication time of τ_k for the assignment SR_z, and $f(X)$ is the function which returns the corresponding communication time if the cost of communication is X. We can assume that the cost of transferring data is equivalent to the transferring time, i.e., $f(X) = X$, thereby Eq. 5 is achieved.

$$t_{commun} = \frac{T_k}{h} \sum_{\forall i, R_i \in \tau_k} \sum_{\forall j, R_j \in R} C_{R_{ij}}(SR_z) \tag{5}$$

- *Cooling Schedule:* There are two common types of cooling schedules, namely, monotonic and non-monotonic. The cooling schedule of both SA and SMSA in this paper is assumed monotonic in the sense that in each iteration, the current temperature is decreased with a constant gradient. In the original monotonic SA, the temperature

in the current iteration is equal to μ × the temperature in the previous iteration, where μ is a real value between zero and one. According to [17], a much more efficient monotonic cooling schedule can be achieved by

$$\psi_i = (\frac{(\psi_s - \psi_f)(b+1)}{b})(\frac{1}{i+1} - 1) + \psi_s,\ i = 1,2,...,b \tag{6}$$

where ψ_s and ψ_f are the start and final temperature respectively, and b is the expected number of observed temperatures which should reflect the inputs parameters that effect on complexity of the problem. We set it equal to $m^2 \times N$. Tuning the value of ψ_s and ψ_f has a strong impact on both the execution time and convergence of SA and SMSA. Clearly, increasing the distance between the start and the final temperature would improve the solution quality, however, this would also increase the running time of the algorithm. Thus, a tradeoff should be made regarding this issue. Based on [18] an appropriate value for the initial and final temperatures of the SA can be achieved by the Eq. 7 and 8 respectively, and we use them in the SMSA.

$$\psi_s = \frac{TC^{best} - TC^{worst}}{\log 0.9} \tag{7}$$

$$\psi_f = \frac{TC^{best} - TC^{worst}}{\log 0.01} \tag{8}$$

where TC^{best} denotes the lower bound of the total cost function in the problem space, and the TC^{worst} indicates the upper bound for this function. It is not difficult to estimate an upper bound and a lower bound for the total cost function. The Eq. 9 creates an upper bound whereas Eq. 10 makes a lower bound for the total cost function.

$$TC^{worst} = \gamma \times \frac{T_{Y_i(SR_z)}}{h} \times \sum_{i=1}^{m} \sum_{j=i+1}^{m} cr_{ij} + \sigma \times \sum_{\forall k, \tau_k \in \tau} \frac{E_k^{max}}{T_k} \tag{9}$$

where E_k^{max} denotes the maximum execution time of the kth task which can be easily calculated by this assumption that the τ_k communicate with other tasks by the cost γ.

$$TC^{best} = \gamma \times \sum_{i=1}^{m} \sum_{j=i+1}^{m} cr_{ij} \tag{10}$$

- *Stopping Condition:* The algorithm terminates when ψ_i becomes less than ψ_f.

The pseudo code of the algorithm is provided in Alg. 1.

The execution time of SMSA is potentially longer than that of SA because, to select a neighbor in SMSA, the total cost function for $(N-1)$ neighbors is computed while when using SA it is computed for one neighbor only. To compensate the slower execution time of SMSA, we suggest a fast method to compute the total cost of each solution that is used instead of the mentioned total cost function (Eq. 1). The following three minor changes can make it significantly faster.

1) According to the above-mentioned definition of neighborhood structure, for each neighbor, only one of tasks moves to another core and the rest of tasks remain on the previous cores. Therefore, for each solution, instead of recomputing the execution time of all tasks, we need to only recompute the execution time of the task which moved, along with the execution time of other tasks communicating with this task. To implement this idea, before starting the SMSA, and after creating the task set τ, for each task τ_i, we create the list L_i which includes the tasks communicating with τ_i including τ_i itself. Accordingly, if in a neighbor, the location of τ_k is changed, then to compute the total cost of that neighbor, only execution time of tasks in the L_k are recomputed, and the execution time of other tasks are the same with the current solution. However, for the initial solution we should still use the Eq. 1 to compute the total cost.
2) We can restrict the range of the second summation in Eq. 5 while the result is still correct. It can be re-written as

$$t_{commun} = \frac{T_k}{h} \sum_{\forall i, R_i \in \tau_k} \sum_{\forall j, R_j \in \tau_l \text{ and } \tau_l \in L_l} C_{R_{ij}}(SR_z) \tag{11}$$

3) The term $C_{R_{ij}}(SR_z)$ is common between Eq. 1 and Eq. 4, and thus we can store it when we are computing the cost function and reuse (instead of re-computing) it in the function for calculating the execution time of tasks.

In this paper, SMSA with the new fast total cost function is called SMSA$^+$. Experimental results in Section V manifest the significant speed-up of SMSA$^+$ in comparison to SMSA.

Algorithm 1 SMSA

```
Inputs: task set τ generated based on the transaction set, and the RIG
Initialize the algorithm parameters ψ_s, ψ_f, Q, and σ
Generate the initial solution SR_0 randomly
TCV_0 = TC(SR_0) {Compute the total cost function for SR_0}
SR_b = SR_c = SR_0 {assign the initial solution to both of the current solution and the best solution}
TCV_b = TCV_c = TCV_0 {assign cost of initial solution to both cost of current and cost of best solution}
Add SR_0 to the queue of recently visited solutions
ψ_c = ψ_s {assign the start temperature to the current temperature}
repeat
    SR_n = Select one of the neighbors based on the stochastic-systematic selection
    if SR_n is not visited recently then
        TCV_n = TC(SR_n)
        Δ = TCV_n − TCV_c
        if Δ ≤ 0 then
            SR_c = SR_n
            TCV_c = TCV_n
            Add SR_n to the queue
            if TCV_n ≤ TCV_b then
                SR_b = SR_n
                TCV_b = TCV_n
            end if
        else
            Generate a uniform random value x in the range (0, 1)
            if x < e^(−Δ/x) then
                SR_c = SR_n
                TCV_c = TCV_n
                Add SR_n to the queue
            end if
        end if
    else
        Update location of SR_n in the queue
    end if
    update ψ_i based on the Eq. 6
until ψ_i ≤ ψ_f
return the SR_b
```

B. Framework 2: The Refinement Approach

In this section the second solution framework is presented. This framework is quite similar to the first framework however

with one major difference. A task set is generated similar to how it is done in the first framework and then we use SMSA to allocate the task set onto the cores. However, a REfinement Function (REF) is defined which attempts to merge the tasks which communicate together, and are allocated on the same core. The basic notion of this function is that if we merge two tasks which communicate together into one task, then the communication cost will be reduced because, communication between these tasks is achieved at a lower cost α instead of the cost of β. It should be mentioned that based on [19], if we merge two tasks, then period of the created task is set to the greatest common divisor (gcd) of periods of those tasks. Therefore, if two tasks with different periods are merged together, then period of the new task may become a significantly lower value, and thus the dedicated utilization of the new task may be significantly higher than the sum of utilization of those two tasks. Accordingly, in order to avoid a large increase of CPU utilization, REF does not merge the tasks with different periods. It is worth noting that if we allow to merge tasks with different periods looking both at the communication cost and the amount of increase of CPU utilization, then the problem is clearly a trade-off between communication and CPU utilization, which is out of the scope of this paper.

The interesting point in merging tasks with the same period is that it does not only reduce communication cost, but it also reduce CPU utilization. This decrease of utilization is derived based on the Eq. 11, because the links between the merged tasks (which are supposed to merge) are turned from the cost β to a lower cost α. In Section V, the capability of the REF to decrease the CPU utilization along with the total communication cost reduction is demonstrated by a large number of experiments. The pseudo code of the REF is presented in Alg. 2. Some further points of the REF algorithm are as follows:

- Two tasks communicate with each other if and only if at least one of the runnables of the first task communicates with one of the runnables of the second task.
- Two tasks are *mergeable* if they are located on the same core, they have the same period, and they communicate with each other.

There are three options to integrate Alg. 2 with SMSA. The first one is that we run REF before SMSA, and the second one is to run REF after SMSA and the last option is to call REF from inside SMSA. The first option restricts the search space and thus it might loose an optimal solution in which due to the configuration of the RIG, two tasks with the same period should be allocated to different cores. Even worse than that is the dependency of the task execution time to the assignment of tasks (based on the Eq. 5) which leads that when the tasks have not still scheduled, the execution time of tasks are unknown, and thereby merging some communicating tasks can result in an unschedulable task set. Accordingly, the first option is not taken into account in this paper. The second option is named SMSA with Non-Feedback Refinement (SMSANFR) where after finishing SMSA, based on the generated assignment it attempts to merge the tasks in order to minimize the communication cost. However, since SMSA is not aware about the task refinement procedure (Not using feedback of the refinement function), it may select another solution as the optimal one. For example, let's suppose that X and Y are two candidates for the solution of the problem and before refinement X outperforms but after refinement due to a stronger merging is applicable on Y, it surpasses. To manage this issue, we adopt the last option where inside SMSA we invoke REF to refine the task set before evaluation of each individual. In this way, SMSA reflects the effect of task merging in guiding the search towards an optimal solution. This algorithm is called SMSA with Feedback Refinement (SMSAFR). To implement this algorithm, it is sufficient to invoke REF at the beginning of the total cost function and then the total cost is computed for the new task set created by REF. In the evaluation section, the higher efficiency of the SMSAFR in comparison to both SMSA$^+$ and SMSANFR is shown. However, the SMSAFR takes a little longer execution time because we call REF several times compared to the first and second options in which a call is made only one time after the scheduling.

TABLE I
APPLICATION PARAMETERS AND THE CORRESPONDING VALUE RANGES.

Parameters	Description	Value ranges
c	communication rate per hyper-period	[0,2000] KB
e	runnable execution time	[2,100] msec
D_i	transaction deadline	[400,1200] msec
$\|\Gamma\|$	number of runnables in each transaction	[1,10]

Algorithm 2 REF

```
1:  Inputs: the task set τ and the SR vector
2:  for each task i do
3:      for each task j do
4:          if τ_i and τ_j are meregeable then
5:              Merge them into the task τ_i
6:              Update tasks' indices
7:              Decrement the number of tasks
8:          end if
9:      end for
10: end for
11: return the updated task set
```

V. PERFORMANCE EVALUATION

In this section the performance of the proposed algorithms is assessed based on a large number of experiments. We created a set of randomly generated applications which are supposed to be executed on a multi-core processor. In Tab. I application parameters along with the corresponding value ranges are mentioned. In addition, two types of multi-core chips with the size of four cores and eight cores respectively are considered as the hardware configuration.

For each problem size, all the algorithms (SA, SMSA, SMSA$^+$, SMSANFR, and SMSAFR) ran 20 times to reach %95 confidence interval. They are run in C# 4.5 and on a PC with 2.2 GHz Intel Core i7 and 6 GB of RAM memory. Furthermore, in order to illustrate the quality of the proposed algorithms, we also implemented an exhaustive approach to generate the exact solution. The exhaustive algorithm is based on the the Back-Tracking (BT) search which traverses a search tree whose leaves correspond to potential solutions to the task

assignment problem. We use a fast bounding method that prunes unpromising branches that cannot lead to an optimal solution. To find out whether a vertex is promising or not, the CPU utilization of all the cores should be computed, and if the CPU utilization of at least one of them is greater than one, then the solution is unpromising, and otherwise it is a promising solution. To compute the CPU utilization we need to calculate the tasks' execution times for each vertex of the search tree. This simple way consumes a long time to compute tasks' execution times for each vertex. A faster and smarter way could be to compute a minimum execution time for each task irrespective of scheduling of tasks, that could be performed before starting the BT algorithm. In order to calculate the minimum task execution time, we assume the cost β for all communication between tasks. It leads to a minimum utilization for each vertex and if the minimum utilization of a core is greater than one, then the actual utilization is definitely equal or higher, and thus the vertex is unpromising. It should be noted that only for the leaves, the total cost function (Eq. 1) is invoked which of course, works with the actual utilization. Since, the execution time of the BT algorithm for large size problem instances is extremely high, we only run it for the small and moderate size of the problem.

TABLE II
ALGORITHM PARAMETERS AND THE CORRESPONDING VALUE RANGES.

Parameters	Description	SA	SMSA
ψ_s	start temp.	by Eq. 7	by Eq. 7
ψ_f	final temp.	by Eq. 8	by Eq. 8
n	expected # of observed temps	-	$m^2 \times N$
μ	cooling factor	0.99	-
Q	size of queue	-	m
n	expected # of observed temps	-	$m^2 \times N$
σ	penalty coefficient	$10 \times Cost^{max}$	
$B_\alpha = \frac{1}{\alpha}$	intra-task bandwidth	$5GB/sec = 5000KB/msec$	
$B_\beta = \frac{1}{\beta}$	inter-task bandwidth	$4GB/sec = 4000KB/msec$	
$B_\gamma = \frac{1}{\gamma}$	inter-core bandwidth	$2GB/sec = 2000KB/msec$	

As we mentioned before the algorithm parameters have a substantial impact on the performance of both SMSA and SA. In order to ensure a fair comparison, we strive to select the best value for the SA parameters. For this purpose, for each parameter all possible values in a reasonable range are checked, and the value which conducts the global optimum is chosen. All algorithm parameters are listed in Tab. II.

The bandwidth values in this table are selected based on the average of the actual bandwidth for some of the common multi-core processors in the market such as Intel, AMD Phenom. However, based on the published specifications by the Intel and AMD, the theoretical bandwidth of inter and intra-core are multiple times more than the mentioned values in the table (around [12,32] GB/sec for the intra-core bandwidth, and [2.8,18] GB/sec for the inter-core bandwidth).

In Tab. III the generated results by SA, SMSA, SMSA$^+$, and BT for running five different applications on a four-core processor are listed. As a minor remark it should be mentioned that the execution times' columns in the table imply the execution time of searching in the problem space (not tasks' execution times). It is noticeable that in all experiments, SMSA outperforms SA in term of both the total communication cost and the execution time, that this preference is becoming significant when the size of problem grows up. In average the SMSA reduces the total communication cost by 17% in comparison to SA. In addition the deviation of SMSA from optimal solution is less than 1% in the experiments. Another interesting point which can be observed by this table is that the execution time of SMSA$^+$ is 32% better than that of SMSA. In order to evaluate the second solution framework, a set of

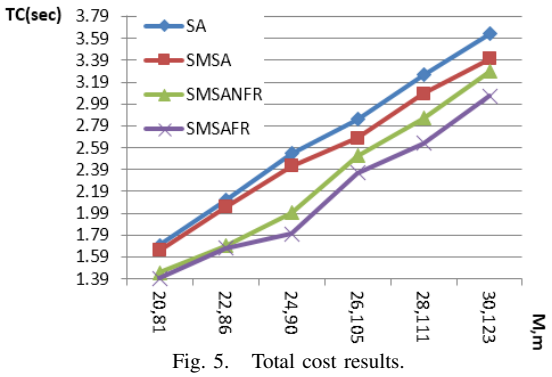

Fig. 5. Total cost results.

larger applications are considered which are supposed to be executed on an eight-core processor. Fig. 5 represents results of the experiments where each horizontal element is a pair of the number of transactions and the number of runnables while vertical elements denotes the sum of inter-runnable communication time. As we already expected SMSAFR surpasses other algorithms. On average SMSAFR decreases the total communication time by 24%, 18%, 6% in comparison to SA, SMSA and SMSANFR respectively. On the other hand, as we already anticipated the execution time of SMSAFR is 14% slower than the SMSANFR on average. As based on our model the processing power of the cores are the same, the only possible way to reduce the average of utilization of the cores is to make the communication time between the runnables lower. Therefore, it is reasonable that SMSAFR generates superior solutions in terms of CPU utilization in comparison to the other algorithms discussed in the paper. On average SMSAFR reduces the average processor utilization by 11%, 10%, 3% in comparison to SA, SMSA and SMSANFR respectively. Fig. 6 illustrates the results where the applications are executing on an eight-core processor.

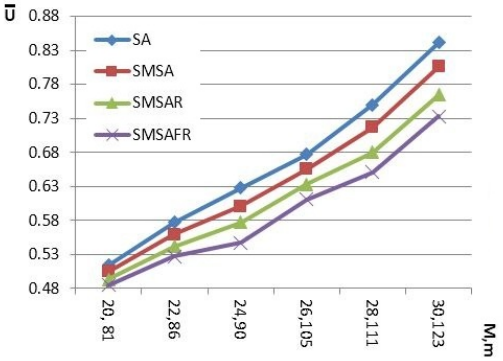

Fig. 6. Average utilization results.

TABLE III
EXPERIMENTAL RESULTS FOR VARIOUS APPLICATIONS RUNNING ON A FOUR-CORE PROCESSOR.

Problem Size		Simulated Annealing		SMSA		Back Tracking	
# of Transactions	# of Runnables	Total Comm. Cost (sec)	Execution Time (sec)	Total Comm. Cost (sec)	Execution Time (sec) SMSA/SMSA$^+$	Total Comm. Cost(sec)	Execution Time (sec)
9	40	1.0832	1.42	1.0832	3.15/2.21	1.0832	16.90
11	44	1.2047	1.78	1.1692	3.41/2.41	1.1692	129.27
13	50	1.3315	1.97	1.3165	3.80/2.64	1.3165	1880.04
14	53	1.3618	2.35	1.3374	5.46/3.59	1.3345	6132.66
15	60	1.4712	2.87	1.4370	6.31/4.23	1.4348	11988.58
average		1.2904	2.078	1.2686	4.42/3.016	1.2674	4029

VI. CONCLUSION

In this paper, we have investigated some challenges related to achieving a resource efficient and predictable design of AUTOSAR software for multi-core real-time systems. Specifically, we have looked into the challenges of designing a resource efficient solution in terms of minimizing the overall communication cost inherent in communication among AUTOSAR runnables executing on a multi-core processor. We have discussed possible solutions addressing this problem, and among the different design options encountered, two solution frameworks have been proposed and explained in detail. The first solution framework tries to achieve a good mapping of runnables to tasks, that in turn are mapped to the cores of a multi-core processor. The second framework tries to, in addition to the mapping of the first framework, also merge tasks on the same core such that the level of inter-task communication is reduced and thereby the overall communication cost will decrease. Furthermore, in this framework, to avoid increasing CPU utilization, only the tasks with the same period are allowed to be merged. There is a third solution that we have left for future work, also allowing a reshuffling of tasks among the cores in order to find a globally minimal communication cost for the whole system. There are additionally two revenues of future work of this paper. Firstly, allowing the merging algorithm to merge tasks with arbitrary periods if the ratio of decreasing the communication cost at the expense of increasing the CPU utilization is higher than a constant coefficient. This constant coefficient should be determined based on an acceptable trade-off between the communication cost and the CPU utilization. The final target for future work is to extend the communication efficient mapping problem setting towards a heterogeneous distributed system in which each inter-connected node could be a multi-core processor.

ACKNOWLEDGMENT

The work presented in this paper is supported by Mälardalen University and Vinnova via the FFI initiative "AUTOSAR for Multicore in Automotive and Automation Industries".

REFERENCES

[1] AUTOSAR methodology, AUTOSAR std. [Online]. Available: http://www.autosar.org

[2] F. Afrati, C. Papadimitriou, and G. Papageorgiou, "Scheduling dags to minimize time and communication," in *VLSI Algorithms and Architectures*. Springer, 1988, pp. 134–138.

[3] R. Mehrotra and S. N. Talukdar, *Scheduling of tasks for distributed processors*. ACM, 1984, vol. 12, no. 3.

[4] P. M. Yomsi and Y. Sorel, "Schedulability analysis for non necessarily harmonic real-time systems with precedence and strict periodicity constraints using the exact number of preemptions and no idle time," in *4th Multidisciplinary International Scheduling Conference*, 2009.

[5] D.-T. Peng and K. G. Shin, "Static allocation of periodic tasks with precedence constraints in distributed real-time systems," *IEEE transaction on software engineering*, vol. 23, pp. 745–758, 1997.

[6] J. M. Rivas, J. J. Gutiérrez, J. C. Palencia, and M. González Harbour, "Schedulability analysis and optimization of heterogeneous edf and fp distributed real-time systems," in *23rd IEEE Euromicro Conference on Real-Time Systems (ECRTS)*, 2011, pp. 195–204.

[7] T. C. Xu, P. Liljeberg, J. Plosila, and H. Tenhunen, "Exploration of heuristic scheduling algorithms for 3d multicore processors," in *15th International Workshop on Software and Compilers for Embedded Systems*. ACM, 2012, pp. 22–31.

[8] Y. Yang, "Software synthesis for distributed embedded systems," Ph.D. dissertation, Ph. D. thesis/Yang Yang, 2012.

[9] W. Peng, H. Li, M. Yao, and Z. Sun, "Deployment optimization for autosar system configuration," in *2nd IEEE International Conference on Computer Engineering and Technology (ICCET), 2010*, vol. 4, 2010, pp. 189–193.

[10] R. Long, H. Li, W. Peng, Y. Zhang, and M. Zhao, "An approach to optimize intra-ecu communication based on mapping of autosar runnable entities," in *IEEE International Conference on Embedded Software and Systems*, 2009, pp. 138–143.

[11] Q. Zhu, H. Zeng, W. Zheng, M. Di Natale, and A. Sangiovanni-Vincentelli, "Optimization of task allocation and priority assignment in hard real-time distributed systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 11, no. 4, p. 85.

[12] M. Zhang and Z. Gu, "Optimization issues in mapping autosar components to distributed multithreaded implementations," in *22nd IEEE International Symposium on Rapid System Prototyping (RSP)*, 2011, pp. 23–29.

[13] E. Wozniak, A. Mehiaoui, C. Mraidha, S. Tucci-Piergiovanni, and S. Gerard, "An optimization approach for the synthesis of autosar architectures," in *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*. IEEE, 2013, pp. 1–10.

[14] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, "Period optimization for hard real-time distributed automotive systems," in *Proceedings of the 44th annual Design Automation Conference*. ACM, 2007, pp. 278–283.

[15] H. R. Faragardi, R. Shojaee, and N. Yazdani, "Reliability-aware task allocation in distributed computing systems using hybrid simulated annealing and tabu search," in *High Performance Computing and Communication 14th IEEE International Conference on*. IEEE, 2012, pp. 1088–1095.

[16] H. R. Faragardi, R. Shojaee, M. A. Keshtkar, and H. Tabani, "Optimal task allocation for maximizing reliability in distributed real-time systems," in *Computer and Information Science, 12th IEEE/ACIS International Conference on*. IEEE, 2013, pp. 513–519.

[17] S. Seyed-Alagheband, H. Davoudpour, S. H. Doulabi, and M. Khatibi, "Using a modified simulated annealing algorithm to minimize makespan in a permutation flow-shop scheduling problem with job deterioration," in *Proceedings of the world congress on engineering and computer science*, vol. 2, 2009, pp. 20–22.

[18] Q.-M. Kang, H. He, H.-M. Song, and R. Deng, "Task allocation for maximizing reliability of distributed computing systems using honeybee mating optimization," *Journal of Systems and Software*, vol. 83, no. 11, pp. 2165–2174, 2010.

[19] H. R. Faragardi, B. Lisper, and T. Nolte, "Towards a communication-efficient mapping of AUTOSAR runnables on multi-cores," in *Emerging Technologies and Factory Automation, 18th IEEE Conference on*. IEEE, 2013, pp. 1–5.