A Communication-Aware Solution Framework for Mapping AUTOSAR Runnables on 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*—An AUTOSAR-based software application contains a set of software components, each of which encapsulates a set of *runnable* entities. In fact, the mission of the system is fulfilled as result of the collaboration between the runnables. Several trends have recently emerged to utilize multi-core technology to run AUTOSAR-based software. Not only the overhead of communication between the runnables is one of the major performance bottlenecks in multi-core processors but it is also the main source of unpredictability in the system. Appropriate mapping of the runnables onto a set of tasks (called mapping process) along with proper allocation of the tasks to processing cores (called task allocation process) can significantly reduce the communication overhead. In this paper, three solutions are suggested, each of which comprises both the mapping and the allocation processes. The goal is to maximize key performance aspects by reducing the overall inter-runnable communication time besides satisfying given timing and precedence constraints. A large number of randomly generated experiments are carried out to demonstrate the efficiency of the proposed solutions.**

***Keywords*-AUTOSAR; runnable; mapping; multi-core; feedback-based search; Simulated Annealing; Ant System.**

I. Introduction

AUTOSAR [1] is applied as a standard architecture to develop automotive embedded software systems. AUTOSAR originated from component-based software engineering where a system is divided into a number of software components each of which encapsulates a set of related functions which are called *runnable* entities in the context of AUTOSAR. As a result of collaboration of these runnables, the mission of the system is fulfilled. Recent trends towards using multi-cores in automotive industry [2] arises new challenges related to how AUTOSAR can utilize the maximum potential of multi-core platforms. Timing predictability and obtaining a reasonable performance are the most important challenges in running an AUTOSAR-based software on a multi-core processor. Such challenges are inherent in 1. the communication between runnables located on different cores, 2. shared resources, where the former is the focus of this paper. Communication time plays a significant role, as it impacts on performance aspects relevant in the domain of embedded real-time systems, such as throughput and response time [3]. In other words, if the overall inter-runnables communication time is reduced then the total utilization of a given workload is decreased and as a result of the utilization reduction, system throughput and response time can be potentially improved.

The runnables should be mapped onto a set of tasks, this process is called *mapping*. The mapping directly affects the schedulability of the task set. The generated task set should then be allocated among the cores of a multi-core processor, this process is called *allocation* (partitioning). To reach an optimal solution both the mapping and allocation must be considered together because, if the task set is not properly formed, even a highly efficient task allocation in terms of communication overhead may not lead to an acceptable solution, and vice versa.

In order to address the problem, it can be formulated as an optimization problem. For this purpose, the communication time between the runnables should be formulated for different scenarios such as inter-core and intra-core communications. The analysis is totally dependent on the memory hierarchy used in multi-core processor architectures. In this paper, an abstract communication model for shared-memory multi-cores is applied where inter-core communication usually happens through the memory sharing mechanism. Based on the communication time analysis the optimization problem is proposed. An efficient heuristic solution to cope with the optimization problem forms the key part of a solution framework. The solution should deal with both the mapping and allocation processes to satisfy all timing and precedence constraints along with minimizing the overall inter-runnable communication time. In this paper, three solutions are suggested which subsequently develop the framework in the sense that the first solution is the simplest one and the third solution applies a more flexible mapping algorithm along with a new parallel evolutionary algorithm.

In order to evaluate the proposed solution framework, a large number of randomly generated experiments are carried out for different sizes of systems. For each system size the three solutions are executed and as we intuitively expected, the third solution outperforms the other two in terms of both assignment quality and execution time. Furthermore, to demonstrate the size of the gap between the assignments generated by the proposed solutions and the optimal assignment, an exhaustive search is executed for small size systems.

Concisely, the main contributions of this paper are:

1) We introduce a solution framework for mapping of AUTOSAR-based software on a multi-core processor, subject to minimizing the overall inter-runnables com-

munication time.
2) Three different solutions are suggested to develop the proposed framework.
3) An abstract communication time analysis for shared-memory multi-core processors is proposed.
4) A novel parallel evolutionary algorithm is introduced to find an efficient allocation of tasks to cores in an acceptable execution time.

The paper is organized as follows: In Section II a brief review of related work is presented. The problem is described in detail and assumptions are defined in Section III. In Section IV, the three solutions are introduced. In Section V the performance of the proposed solutions are compared. Finally, the main results are concisely summarized along with directions for future work in Section VI.

II. Related Work

A large number of studies have been conducted to solve the challenges related to static allocation of communicating real-time tasks to a set of processors [4], [5], [6]. In such studies, the task set is often described as an acyclic directed graph where the tasks indicate nodes, and the edges between the tasks display either data dependency [7] or triggering [8]. In [9] a holistic solution containing task allocation, processor scheduling and network scheduling was presented. They applied Simulated Annealing (SA) to find an optimal task allocation in a heterogeneous distributed system. 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 the assigned tasks on each processor. Due to the exponential nature of BB, it fails in finding a solution for most real-world sized problems. In [10] this problem is solved for a more general communication model where the tasks can send data to each other at any point of their execution – not necessarily at the end point of their life time. In [11] a mixed linear integer programming formulation is proposed to model the problem where in addition to find a proper allocation of tasks to the processing nodes, task priorities and the minimum cost of underlying hardware are taken into account. Similarly, [7] investigated the problem of task allocation, priority assignment and signal to message mapping however, the goal in the paper is to minimize end to end latencies.

The above-mentioned studies have been dedicated to the task allocation problem on a distributed system however, there are few works which are considering the problem on a multi-core system. Although the proposed solutions are more general and may be applicable for multi-core systems, focusing on inter-core communication properties can lead to more efficiency and higher predictability. The approaches introduced for real-time multi-core systems [12], [13], [14] cannot be directly applied in the context of AUTOSAR systems because a proposed solution should be able to embrace the problem of mapping runnables into a task set along with allocation of the task set onto the cores. An improper task set even with an optimal task allocation may not result in a reasonable system performance. Recently, in [15] the problem of assigning a set of runnables into tasks is taken into consideration where the target objective is to minimize the end-to-end latencies. The authors apply a genetic algorithm to cope with the problem. However, unfortunately it is often not possible to obtain optimal solutions (or even feasible solutions) if the two activities (mapping and allocation) are handled in isolation, similar to what they do.

III. Problem Modeling

In this section, first the problem is formally defined, then a communication time analysis is suggested considering the architecture of the target processor. Finally the problem is formulated as an optimization problem.

A. Problem description

Let's suppose that there is a set of loosely-coupled runnables which could be concurrently executed on different cores of a multi-core processor. 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 cores, $\rho = \{\rho_j : j = 1,2,...,N\}$ of a homogeneous multi-core chip. The runnable R_i has a Worst Case Execution Time (WCET) denoted by e_i. Runnables have inter-communication relationships that are assumed to be based on non-blocking read/write semantics [16]. Let's suppose that runnables have read-execute-write semantic, meaning that they first read input data, then perform manipulations and finally write output data. In our model two types of communication between the runnables are taken into account. The first type covers data dependency between the runnables where they have to run with a specific execution order to fulfill the dependency whereas, the second communication type is when a pair of runnables can transfer data in between each other, while there may not be any precedence among their execution order (e.g., there is a shared buffer with sufficient size).

The first communication type is modeled by a set of independent transactions $\{\Gamma_i : i = 1,2,...,M\}$ where each of which represents an end-to-end function implemented by a sequence of runnables. Indeed, each transaction is a directed acyclic graph in which each node is a runnable and links represent data dependency between them. It should be noted that the dependency between the runnables in a transaction does not imply triggering, in the sense that a successor can start with obsolete data generated by its predecessor. However, to fulfill the mission of a transaction all successors must start to run with the fresh data. Fig. 1 shows a sample of a transaction. 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 covering this runnable.

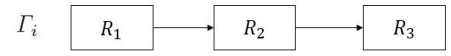

Fig. 1. A sample of a transaction.

The transaction Γ_i has a relative end-to-end deadline denoted by D_i before which all runnables of the transaction

must finish their execution. The transaction deadline is corresponding to the deadline of the mission of that transaction. For example, the mission could be the braking system in a car where the whole transaction must be performed before a specific end-to-end deadline. There are three approaches to handle the scheduling of such transactions, time triggering, event triggering and mixed time/event triggering. In this paper, we adopt the time triggering approach in the sense that a transaction arrives periodically or sporadically, but with a known minimum inter-arrival time denoted by P_i. In the time triggering approach, determining the optimal period of transactions strongly affects the system performance [16] because, finding a maximum period in which a transaction meets its deadline reduces processing load. Nevertheless, determining the optimal periods is not included in the scope of this paper. Instead, a conservative approach is to consider the period of a transaction equal to its given relative deadline. It is worth noting that all runnables in a given transaction share the same period which is equal to the transaction period.

The second communication type is modeled by an undirected graph called Runnables Interaction Graph (RIG). Each node of the RIG represents a runnable and the arcs between the runnables show data communication between them. Furthermore, there is a label on each arc which indicates the amount of data that should be transferred between a pair of runnables in both directions per *hyper-period*. The hyper-period is the Least Common Multiple (LCM) of the periods of all the transactions, denoted by H. When a transaction has a short period then it generates data more frequently, hence, to compare the amount of data sent across various runnables irrespective of their periods, we consider data transfer rate per hyper-period. Fig. 2 illustrates a RIG instance.

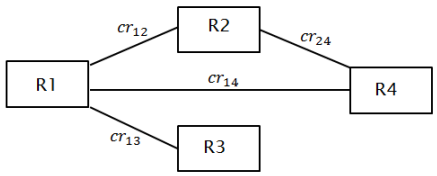

Fig. 2. A sample of a RIG in an AUTOSAR system.

B. Communication time analysis

Let us look more closely at the target multi-core architecture to analyze different communication time delays. Let's suppose a multi-core processor with a common three-level cache architecture. The shared-cache architecture has become increasingly popular in a wide variety of embedded real-time applications [17]. In such architectures each core has its own private $L1$ cache while a second-level ($L2$) cache is shared across each pair of cores, and finally a third-level cache (L3) is shared among all cores. It is difficult to characterize the latency values with precise numbers, but in general L2 cache latency is almost two to three times larger than the L1 cache latency, L3 cache latency is roughly ten times larger than L1 cache latency, and RAM latency is two orders of magnitude larger than the latency of the L1 cache [18]. Fig. 3 represents a sample of such an architecture with 4 processing cores.

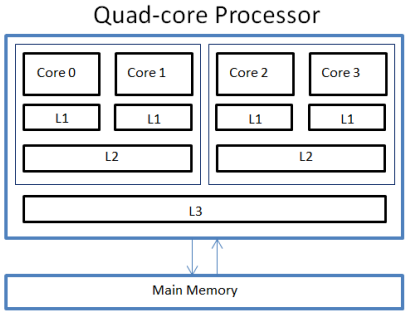

Fig. 3. A three-level shared-cache quad core architecture.

In the abstract model, four scenarios to communicate between the runnables are considered.

I. If the runnables R_i and R_j are allocated within the same task, then the runnables share the same address space and can communicate to each other through the local cache (L1) even without requiring a context switch. It should be noted that in the same task, since the time interval between writing and reading data is too short, the chance of preemption and removal of data from the L1 cache during this period is negligible. Let's suppose that the worst-case latency to transfer each unit of data in this scenario is α.

II. When the runnables are allocated in different tasks on the same core, a first-level cache miss is more probably to happen in comparison to the first scenario. The reason is that after finishing the execution of a writer task, the scheduler may select another task to run instead of the reader task. It also imposes a context switch overhead twice. Let's suppose that the worst-case latency to transfer each unit of data in this scenario is β.

III. When the runnables are executed on separate cores which share a L2 cache, the communication can go through the second-level cache. In this scenario, the worst-case communication delay for each unit of data is θ.

IV. Finally, when they are located on different cores without a shared L2 cache, communication has to go through the shared memory (or L3 cache). As the shared memory has a significantly larger latency than the local cache, a considerable difference is expected between this and the third scenario [3]. γ is chosen to represent the worst-case delay in this case.

It is interesting to note that this model can easily be generalized to cope with other common types of shared-cache processors. For example, if in the processor architecture, the L2 cache is also private for each core and it is not shared among pair of cores (e.g., Intel Core i7), then it is sufficient to set θ equal to γ. In this case, we expect a lower value for both α and β. Formally, Eq. 1 formulates the above mentioned communication time delays.

$$C_{R_{ij}} = \begin{cases} \alpha \times cr_{ij} & \text{if } I \\ \beta \times cr_{ij} & \text{else if } II \\ \theta \times cr_{ij} & \text{else if } III \\ \gamma \times cr_{ij} & \text{else} \end{cases} \quad (1)$$

where $C_{R_{ij}}$ denotes the delay of data transfer, and I denotes a condition in which R_i and R_j belong to the same task (the first scenario) whereas, II denotes a condition in which the corresponding tasks of R_i and R_j are located on the same core (the second scenario), and III is corresponding to the third scenario.

It is worth to noting that if we want to include more details, then several other factors impact on the communication time analysis such as the size of the first, second and third-level cache, the cache replacement mechanism, the hit rate of cache levels (L1, L2 and L3) etc. Nonetheless, in this paper an abstract communication model is applied to address the problem in a general manner without confining the model to a limited range of shared-cache multi-core processors.

C. Optimization problem

In this subsection, we introduce an optimization problem. There are two variable vectors in the optimization problem, the first vector shows mapping of runnables to tasks and the second one is used for allocation of tasks to cores. Let's suppose that the assignment SR comprises both of these vectors. The straightforward way to model an optimization problem is to define a total cost function reflecting the goal function along with constraint functions. In this case, minimizing the total cost function is equal to minimizing the goal function while there is not any constraint violation. The total cost function for our problem can be computed by Eq. 2 which returns a real value for the assignment SR_z, and this value is used to evaluate the quality of a given assignment.

$$TC(SR_z) = \sum_{i=1}^{m} \sum_{j=i+1}^{m} \frac{T_{Y_i(SR_z)}}{H} \times C_{R_{ij}}(SR_z) + \sigma \times P(SR_z) \quad (2)$$

where $Y_i(SR_z)$ is a function that returns the index of the task to which R_i is mapped by the assignment SR_z, and $P(SR_z)$ is the penalty function being applied to measure satisfiability of a given assignment. It means that if the value of the penalty function is zero, then the assignment SR_z satisfies all the end-to-end timing constraints. 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 the range of the cost function and the importance of the constraint violation. For example, in a soft real-time system, where missing a small number of deadlines may be tolerable, the coefficient should be set to a lower value.

The penalty function should be defined according to the processor scheduler. Because of resource efficiency, most automotive systems are designed based on a static priority-based scheduling [7] using for example, rate monotonic [19] which is also supported by the AUTOSAR standard whereas, dynamic priority schedulers like the Earliest Deadline First (EDF) are not supported by AUTOSAR. Therefore, we assume that each processor employ a preemptive static priority scheduling based on the rate monotonic priority assignment. The penalty function is given by Eq. 3.

$$\begin{aligned} P(SR_z) &= \sum_{i=1}^{N} \sum_{\forall \tau_k, \text{ allocated to the } \rho_i} max\{0, r_k - T_k\} \\ r_k &= E_k(SR_z) + \sum_{j \in hp(k)} \lceil \frac{r_k}{T_j} \rceil E_j(SR_z) \end{aligned} \quad (3)$$

where T_k denotes the period of the task τ_k, and $hp(k)$ implies the tasks with a shorter period than that of τ_k, and $E_k(SR_z)$ indicates the WCET of the kth task for the assignment SR_z which means that in our model task execution time is dependent on both the assignment of runnables to tasks and tasks to cores that will be examined in Section IV-C.

IV. Solution Framework

As we already mentioned, the solution framework should deal with both the mapping and allocation processes. These processes can be done either separately in a subsequent manner, denoted non-feedback-based approach, or carried out together denoted feedback-based approach. In this paper, we propose three solutions where the first one follows the non-feedback-based approach whereas the feed-back based idea is applied to design the other two solutions. The framework is subsequently developed in the sense that the solutions attempts to resolve the disadvantages that exist in the previous solution.

A. Solution 1: Simple mapping

In the first solution, the mapping function is simply carried out by considering each transaction as one task. Consequently, the task deadline is set equal to the transaction deadline, and the task period is also equal to the transaction period (i.e., $P_i = T_i$). In this case, if a task meets its deadline then the corresponding transaction meets its end-to-end deadline as well. In this solution, after creating a task set as the output of the mapping phase, the allocation phase is executed to assign the generated task set among the cores. An evolutionary algorithm called Systematic Memory Based Simulated Annealing (SMSA) [20] is applied as an allocation algorithm. The experimental results in [21] demonstrated that SMSA outperforms SA in the task allocation problem. After finishing the allocation phase, a REfinement Function (REF) is applied on the generated allocation which attempts to merge the communicating tasks located on the same core together. Concisely, REF merges all the mergeable tasks described as follows:

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

The basic notion of REF is that when we merge communicating tasks having equal period time, the overall communication time can be decreased as the communication between these tasks is performed at a lower latency α instead of β. On the other hand, if we merge two tasks with different periods

(deadlines), then to ensure the fulfillment of timing constraints, the deadline of the new task should be set to the minimum deadline of those tasks [22] thus, the period of the new task becomes equal to the minimum period of the merged tasks (remember we assume that the period of each task is equal to its deadline). As a result of this period reduction, not only the total amount of communication data is increased due to sending data more frequently but also the utilization of the new task might be significantly higher than the sum of utilization of those two tasks. Accordingly, in order to avoid the increase of CPU utilization as well as communication time, REF does not merge tasks with different periods.

B. Solution 2: SMSAFR

The second solution is similar to the first solution however with one major difference. In the first solution, since the allocation phase is not aware of the task refinement procedure (not using feedback of the refinement function), it may select a non-optimal solution. For example, let's suppose that X and Y are two candidates for the solution of the allocation problem. Before doing the refinement, X outperforms Y but after refinement, due to a stronger merging applicable on Y, it surpasses. To manage this issue, a feedback-based approach is taken into account by the second solution in which REF is frequently invoked from the inside of SMSA to refine the task set before evaluation of each individual (candidate solution). 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 to refine the task set and then the total cost value can be computed. Although, the SMSAFR is more efficient in comparison to the simple mapping framework in terms of the overall communication time reduction, it takes a longer execution time to search the problem space. The longer search time is inherent in the frequency of the REF invocation. In other words, REF is frequently invoked by SMSAFR compared to the first solution that invokes only one instance of REF after finishing the allocation phase. To address both performance –with respect to optimizing communication time– and framework execution time, a novel third solution is proposed.

C. Solution 3: The utilization-based refinement approach

The main notion in the third solution –PUBRF– is similar to SMSAFR with two principal differences. The first difference is that SMSAFR uses REF as the refinement function which only merges communicating tasks located on the same core with the same period into one task whereas PUBRF uses an extended version of the refinement function called Utilization-Based Refinement (UBR) which merges the tasks on the same core if and only if merging them into one task causes a processor utilization reduction on that core. In other words, UBR considers a trade-off between the amount of inter-task communication and the amount of increasing CPU utilization caused by merging tasks with different periods. Therefore, not only UBR merges the communicating tasks with the same period similar to REF, but it also allows to merge tasks with different periods when they have a lot of communication together. To form this trade-off, the communication time between the tasks should be translated into CPU utilization. We can then easily decide whether merging a pair of tasks is efficient or not in terms of CPU utilization. Eq. 4 integrates communication time and computation time to calculate the utilization of tasks. As the communication time depends on the assignment of tasks to cores, in this equation task utilization is a function of the assignment SR_z.

$$u_{\tau_k}(SR_z) = \frac{E_k(SR_z)}{T_k} \qquad E_k(SR_z) = t_{comput}(k) + t_{commun}(k) \tag{4}$$

where t_{comput} implies the computation time of τ_k which is independent from the assignment of tasks to cores and it can be calculated by

$$t_{comput}(k) = \sum_{\forall l, R_l \in \tau_k} e_l \tag{5}$$

t_{commun} represents the communication time between τ_k and other tasks according to the assignment SR_z which is derived by

$$t_{commun}(k) = \frac{T_k}{H} \sum_{\forall i, R_i \in \tau_k} \sum_{\forall j, R_j \in R} C_{R_{ij}}(SR_z) \tag{6}$$

Another advantage of such a refinement method is that it guarantees if a task set is schedulable on a multi-core processor, after running this refinement function the system keeps its schedulability. In Alg. 1 the pseudo-code of the new refinement function is provided. It should be mentioned that $u_{\tau_{ij}}$ in this algorithm indicates the utilization of the task containing both τ_i and τ_j if we merge them into one task which can be computed by Eq. 7.

$$u_{\tau_{ij}}(SR_z) = \frac{\sum_{\forall l, R_l \in \tau_i \cup \tau_j} e_l + \frac{T_{ij}}{H} \sum_{\forall k, R_k \in \tau_i \cup \tau_j} \sum_{\forall l, R_l \in R} C_{R_{kl}}(SR_z)}{T_{ij}} \qquad T_{ij} = \min\{T_i, T_j\} \tag{7}$$

It is important to emphasize that since we never split a transaction into more than one task, the precedence relations between the runnables of a transaction do not generate precedence constraints between the tasks, and the generated task set is independent.

The second difference is that PUBRF utilizes a more effective evolutionary algorithm which is able to find high quality solutions in a shorter execution time. This evolutionary algorithm is a Max-Min Ant System [23] being leveraged by an asynchronous parallel version of the SMSA algorithm. A flowchart scheme of this framework is given by Fig. 4. As is seen in the flowchart, most parts of this algorithm are implemented in a multi-threading manner where the threads can be executed concurrently on different cores, and thus we exploit the potential of multi-core processors to accomplish a highly efficient search. Some further details regarding the

Algorithm 1 UBR

Inputs: a given task set τ and a vector indicating allocation of the tasks to cores
Create a strictly upper triangular matrix with the size of M, named profitMatrix
for each task i **do**
  for each task $j > i$ **do**
    if τ_i and τ_j are located on the same core **then**
      profitMatrix$[i,j] = \min\{0, (u_{\tau_i} + u_{\tau_j}) - u_{\tau_{ij}}\}$
    else
      profitMatrix$[i,j] = 0$
    end if
  end for
end for
repeat
  Find the pair of tasks with the maximum profit value (let's suppose they are k and l while $k < l$)
  Add all the runnables of the lth task to the kth task
  Set the deleted flag of the lth task (logically remove)
  $T_k = \min\{T_l, T_k\}$
  Recompute the U_{τ_k}
  Set all elements of the lth row and the lth column of the profitMatrix equal to zero
  Recompute non-zero entries of the kth row and the kth column
until (The maximum element of the profitMatrix becomes zero)
Delete the tasks which have been logically removed
return the updated task set

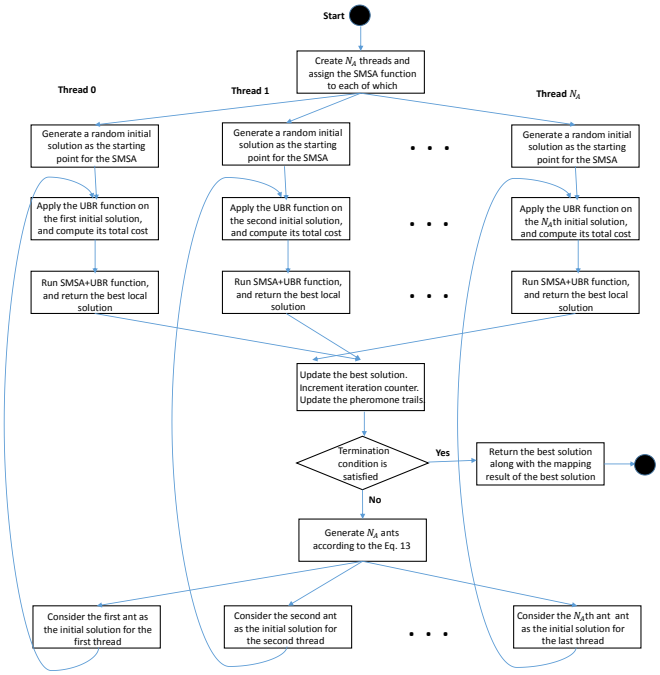

Fig. 4. Flowchart representation for the proposed solution framework.

framework configuration are described below:

- *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 affects the algorithm performance. We represent each allocation solution with a vector of N_{task} elements, and each element is an integer value between one and N. Since the number of tasks in the initial task set is equal to the number of transactions, then $N_{task} = M$. The vector is called Allocation Representation (AR). Fig. 5 shows an illustrative example for an allocation 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*, meaning 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. 5. Representation for assigning the tasks onto the cores.

- *Neighborhood structure used by the SMSA function:* SMSA constitutes a sub set of the problem space that is reachable by moving any single task to any other processing core as the neighbors of the current solution. Therefore, each solution has $M(N-1)$ different neighbors, because each task can run on one of the other $N-1$ cores.
- *Selecting neighbor in SMSA:* 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.
- *Cooling schedule in SMSA:* There are two common types of cooling schedules, namely, monotonic and non-monotonic. The cooling schedule of the SMSA in this paper is assumed monotonic in the sense that the temperature of the current iteration is equal to $\mu \times$ the temperature in the previous iteration, where μ is a real value between zero and one. Based on [24] an appropriate value for the initial and final temperatures of the SA can be achieved by Eq. 8 and 9 respectively, and we use them for SMSA as well.

$$\psi_s = \frac{TC^{best} - TC^{worst}}{\log 0.9} \tag{8}$$

$$\psi_f = \frac{TC^{best} - TC^{worst}}{\log 0.01} \tag{9}$$

where TC^{best} denotes the lower bound of the total cost function in the problem space, and 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. Eq. 10 creates an upper bound whereas Eq. 11 makes a lower bound for the total cost function.

$$TC^{worst} = \gamma \times \frac{T_{Y_i(SR_z)}}{H} \times \sum_{i=1}^{m} \sum_{j=i+1}^{m} cr_{ij} + \sigma \times \sum_{\forall k, \tau_k \in \tau} \frac{E_k^{max}}{T_k} \tag{10}$$

where E_k^{max} denotes the maximum execution time of the kth task which can be calculated by the assumption that

τ_k communicate with other tasks by the cost γ.

$$TC^{best} = \alpha \times \frac{T_{Y_i(SR_z)}}{H} \times \sum_{i=1}^{m} \sum_{j=i+1}^{m} cr_{ij} \qquad (11)$$

It should be mentioned that these values are applied in the first iteration of the main algorithm however, in order to make the main algorithm to converge, the initial temperature in the next iteration of the main algorithm is set to a lower value which is equal to half of the initial temperature in the previous iteration. In other words, when SMSA starts its execution in the first iteration of the main algorithm with the initial temperature ψ_s, then SMSA will start in the second iteration with the initial temperature $\frac{\psi_s}{2}$.

- *Stopping condition of SMSA:* The algorithm terminates when the current temperature ψ_i becomes less than ψ_f.
- *SMSA+UBR:* In the forth level of the flowchart SMSA+UBR implies the SMSA function in which before computing the total cost value, the UBR function is invoked to create a new task set and then the total cost value is calculated for the new task set.
- *Updating the pheromone trails:* Real ants use trails of a chemical substance to communicate with other ants to inform them about the directions in which food can be found. Actually, the pheromone trails are a kind of distributed numeric information which is modified by the ants to reflect their experience achieved during solving a particular problem. In order to apply the ant system to task allocation problems, a pheromone matrix Ph with the size of $M \times N$ is required where the element Ph_{ij} is corresponding to the assignment of the ith task to the jth core. Updating the pheromone trails is done first by lowering the pheromone trails by a constant factor (called evaporation) and then by allowing the best ant to deposit pheromone on the direction that it has visited (called reinforcement). In particular, the update can be performed by

$$Ph_{ij}^{I+1} = \varpi \times Ph_{ij}^{I} + \frac{x_{ij}^{best}}{TC(AR^{best})} \qquad (12)$$

where ϖ denotes the evaporation factor, Ph_{ij}^{I+1} indicates the pheromone value for the next iteration, x_{ij}^{best} is a binary variable which is equal to one if in the best solution the ith task is assigned to the jth core, otherwise it is set to zero, and $TC(AR^{best})$ denotes the total cost value for the best solution.

- *Generation of ants:* The ants are created based on a probabilistic decision relevant to the pheromone values. In other words, if the pheromone value for the element Ph_{ij} is a large value then the ith task will probably be assigned to the jth core in the next ants. This concept is reflected by the following formula

$$Prob^{k}(x_{ij}) = \frac{Ph_{ij}}{\sum_{l=1}^{N_A} Ph_{il}} \qquad (13)$$

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
$\lvert\Gamma\rvert$	number of runnables per transaction	[1,10]

where $Prob^{k}(x_{ij})$ denotes the probability of assigning the ith task to the jth core in the kth ant.

- *Stopping condition of the main algorithm:* The algorithm terminates after a specific number of iterations, denoted by υ.

It is worth noting that the algorithm is developed in such a way that only some light instructions are located in the non-parallel part such as updating the pheromone trails and selecting the next generation of ants while the CPU-intensive functions like the total cost and the UBR are handled in the parallel part. In Section V, it is discussed why in spite of the more complexity of the PUBRF, it surpasses even in terms of search execution time.

V. PERFORMANCE EVALUATION

In this section the performance of the proposed solutions are 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 platform. In Tab. I application parameters along with the corresponding value ranges are mentioned. In addition, two types of multi-core processors are considered where the first one is a quad-core processor and the second one is a six-core processor, both of them have a three-level cache architecture similar to Fig. 3.

For each problem size, all three frameworks were run 20 times to reach 95% confidence interval. They are run in C# 4.5 and on a PC with 2.3 GHz six-core Intel Core i7 and 8 GB of RAM memory. Furthermore, in order to investigate the quality of the proposed algorithms, we also implemented an exhaustive approach to generate the exact solution. The exhaustive algorithm is based on 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 the 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 inter-task communication. 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. 2) is invoked which of course works with the actual utilization. Moreover, to have a fair comparison, the BT algorithm must be equipped with a refinement function in order to allow it to merge the communicating tasks located on the same core. We applied the UBR function to merge tasks before invoking the total cost function in the leaf nodes. Another method could be to consider all possible combinations to merge the communicating tasks located on the same core however, it is extremely time consuming. Even for the current version of BT which is using UBR, we could only run relatively small sizes of the problem.

TABLE II
ALGORITHM PARAMETERS AND THE CORRESPONDING VALUE RANGES

Parameters	Description	Value
ψ_s	start temp.	by Eq. 8
ψ_f	final temp.	by Eq. 9
μ	cooling factor	0.9/0.995
Q	size of queue	m
ϖ	evaporation factor	0.9
N_A	the number of ants	4
υ	the number of iterations	10
σ	penalty coefficient	$10 \times Cost^{max}$
α	intra-task latency	$50\mu sec$
β	inter-task latency	$55\mu sec$
θ	inter-core latency (through L2)	$80\mu sec$
γ	inter-core latency	$85\mu sec$

The algorithm parameters have a substantial impact on the performance of both SMSA and the Max-Min Ant Optimization algorithm. We strive to check all possible values in a reasonable range for every parameter to select the best value for them. Algorithm parameters are listed in Tab. II. The only point related to parameters which may need more clarification is that the cooling factor of SMSA is chosen 0.995 while in the third solution it is set to 0.9 – it is the main reason for the speedup of the third solution in comparison to the first and the second solution. However, the two first solutions are unfortunately not working properly with a cooling factor lower than 0.995.

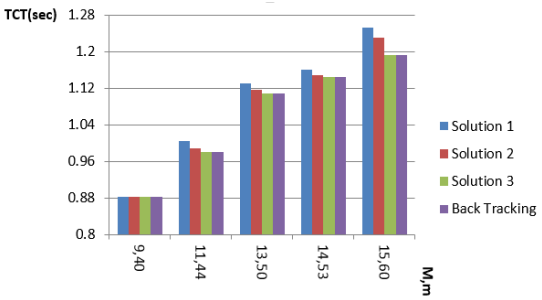

Fig. 6. Average utilization results.

In order to measure the latency values we performed some empirical experiments on a 32-bit version of the Ubuntu 12.04 LTS operating system (kernel version 3.2.29) patched with the PREEMPT RT patch (version 3.2.29-rt44) platform to transfer 1KB data in different communication scenarios (intra-task, itra-task, inter core through L2 and inter-core). To investigate the first scenario we created two functions in a Posix thread. The first function writes 1KB data in a shared array and then the second function is invoked and read the data from the array. Both the writing and reading are done by the sequential data access pattern. Other scenarios were implemented through an event triggering task in which when the first task wrote data on the shared memory, the second task is triggered to read the data. Additionally, to reduce the probability of unwanted interference we put the tasks in the highest priority level.

As the BT algorithm has a very long execution time (around 70 hours for 15 transactions on four cores), we only run it for five small applications on a quad-core platform. In Fig. 6 the results generated by the three solutions along with the BT outputs are listed where horizontal elements are pairs of the number of transactions and the number of runnables while vertical elements indicate the sum of inter-runnable communication time in a hyper period [1]. It is noticeable that in all experiments the results of the third framework and the BT algorithm are identical except to the last experiment where the deviation is still infinitesimal. In order to asses the

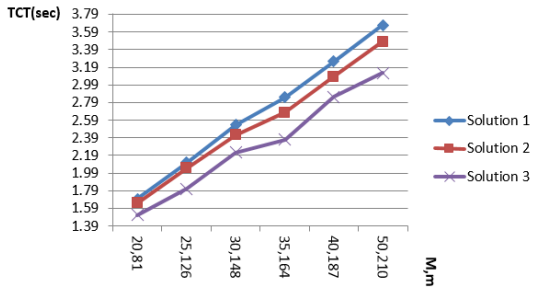

Fig. 7. Total communication time results.

performance of the solutions for larger problem sizes, a set of larger applications is considered which is supposed to be executed on an six-core processor. Fig. 7 represents results of the experiments. As we already anticipated, the third solution outperforms the Simple Mapping and SMSAFR in terms of both the total communication time and the time complexity of the solution. This preference becomes significant with growth of the size of the problem. In average, the third solution decreases the total communication time by 15% and 10% in comparison to the Simple Mapping and SMSAFR respectively. On the other side, since according to our model, task execution time is related to the inter-runnable communication time, it is expected that the third solution is superior in terms of CPU utilization in comparison to the other two frameworks. Fig. 8 illustrates the utilization results. In average the third solution reduces the average processor utilization by 10% and 6% in comparison to the first and the second solution respectively. We also expect that by increasing the size of the application on a higher number of cores the advantage of the third solution becomes more considerable which also indicates scalability of the third solution.

[1]Although GA has been recently applied in the literature, we have not compared our solution with GA because it was already demonstrated in [20] that SMSA outperforms GA for the task allocation problem.

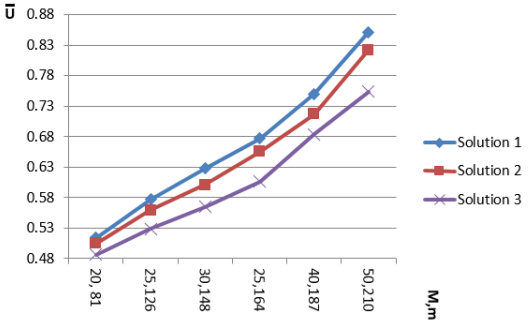

Fig. 8. Average utilization results.

VI. Conclusion

In this paper, we have investigated challenges related to utilization of multi-core platforms in the design of highly efficient and predictable AUTOSAR-based software application. Specifically, we have looked into the challenges of designing a resource efficient solution in terms of minimizing the overall communication time inherent in communication among AUTOSAR runnables executing on a multi-core processor. An abstract communication time analysis was introduced which is able to cover most of the common multi-core architectures. Three solutions were proposed to deal with the problem. Although the second has a better performance than the first one, it often does not yield the best results even for small applications. Therefore, the third solution was suggested which significantly outperforms the other two solutions. The reason to include the first two solutions in this paper (not only the third solution) is that the first solution conforms with the common approach in the literature where the mapping and allocation is done separately, and the second solution implies when we neglect the effect of communication time on CPU utilization. In the future, we plan to investigate another solution which is able to split a transaction into more than one task to handle very large transactions. In addition, transactions with shared runnables could be another direction for future work.

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] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Surveys (CSUR)*, vol. 43, no. 4, p. 35, 2011.

[3] J. Feljan and J. Carlson, "The impact of intra-core and inter-core task communication on architectural analysis of multicore embedded systems," in *ICSEA 2013, The Eighth International Conference on Software Engineering Advances*, 2013, pp. 402–407.

[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] 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.

[8] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and microprogramming*, vol. 40, no. 2, pp. 117–134, 1994.

[9] K. W. Tindell, A. Burns, and A. J. Wellings, "Allocating hard real-time tasks: an np-hard problem made easy," *Real-Time Systems*, vol. 4, no. 2, pp. 145–165, 1992.

[10] J.-M. Chang and M. Pedram, "Codex-dp: co-design of communicating systems using dynamic programming," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 19, no. 7, pp. 732–744, 2000.

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

[12] L. D. Briceño, J. Smith, H. J. Siegel, A. A. Maciejewski, P. Maxwell, R. Wakefield, A. Al-Qawasmeh, R. C. Chiang, and J. Li, "Robust static resource allocation of dags in a heterogeneous multicore system," *Journal of Parallel and Distributed Computing*, vol. 73, no. 12, pp. 1705–1717, 2013.

[13] A. Saifullah, J. Li, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," *Real-Time Systems*, vol. 49, no. 4, pp. 404–435, 2013.

[14] 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.

[15] 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.

[16] 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.

[17] Software techniques for shared-cache multi-core systems. [Online]. Available: https://software.intel.com/en-us/articles/software-techniques-for-shared-cache-multi-core-systems

[18] Performance analysis guide for intel core i7 processor and intel xeon 5500 processors. [Online]. Available: http://software.intel.com/sites/products/collateral/hpc/vtune/performance analysis guide.pdf

[19] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[20] 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.

[21] H. R. Faragardi, K. Sandstrom, B. Lisper, and T. Nolte, "Communication-aware scheduling of autosar runnables on multi-core systems," in *International Workshop on Design Space Exploration of Cyber-physical*. Springer, 2014.

[22] 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.

[23] T. Stützle and H. H. Hoos, "Max–min ant system," *Future generation computer systems*, vol. 16, no. 8, pp. 889–914, 2000.

[24] 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.