Independent Temporal Integration of ARINC653 Conformed Architecture – A Search Based Solution

KUI ZHANG, JUNCHEN LIU, JIAN REN, JINGHUI HU, AND CHAO LIU

1 State Key Laboratory of Software Development Environment, School of Computer Science and Engineering, Beihang University, Beijing 100083, China
2 Aeronautical Key Laboratory for Digital Manufacturing Technology, AVIC Manufacturing Technology Institute, Beijing 100024, China

Corresponding author: Jian Ren (renjian@buaa.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61602021, and in part by the State Key Laboratory of Software Development Environment under Grant SKLSDE-2019ZX-10.

ABSTRACT ARINC653-based integrated modular avionics (IMA) architecture has been widely adopted in the design of modern civil and military aircraft. IMA imposes various requirements on the underlying operating system, of which the temporal and spatial separation requirements are essential to task allocation. In practice, finding the optimal allocation configurations of tasks to enable processing modules to satisfy various temporal constraints is one of the greatest challenges. For that purpose, hundreds of tasks must be mapped into given processing modules, which has been proven to be a nonpolynomial problem. This paper introduces a search-based approach to aid in finding effective solutions for the task allocation problem in polynomial time. Two search techniques based on both population search (genetic algorithm) and neighbor search (simulated annealing), along with their multicore versions, are presented. A heuristic is designed specifically to validate whether candidate solutions fulfill various constraints implied by IMA, and thus to evaluate the fitness. Furthermore, the multicore version is designed to reduce the time delay of obtaining a new optimized configuration. The results show that both algorithms can ultimately find optimized solutions with utility rates above 90% in all configurations and can support the optimization over 100 tasks, which is an outstanding result. The result also reveals that simulated annealing can produce a better solution under limited resources, while the genetic algorithm will determine a valid solution within a shorter time period. Moreover, simulated annealing outperforms the genetic algorithm in terms of both effectiveness and efficiency with respect to solving this allocation problem with complicated constraints.

INDEX TERMS Temporal allocation, Heuristic model, IMA, system integration.

I. INTRODUCTION

In recent years, software integration has received much attention from both industry and research institutions. During the integration process, many attributes must be taken into consideration, such as interface compatibility, coherence of the context, reliability, and temporal performance [1]–[3]. The temporal attributes represent a crucial requirement that must be satisfied in a real-time system, since the correctness of output is not only determined by the result but also depends on the time the result was generated [4]–[6].

Avionics represents one of the most important types of real-time systems, which is now becoming an obligatory component of modern aircraft [7]. Avionics have been used in many applications of aircraft, such as flight control [8], autopilot, collision-avoidance [9], weather systems used in civil aircraft, weapons systems, electronic support measures (ESM) and defensive aids systems (DAS) in the military, etc.

With the requirement of producing better performance while reducing weight, IMA architecture standardized by ARINC653 has been proposed and widely adopted in the design of modern avionics. It was first presented by Honeywell as an Airplane Information Management System (AIMS) on the Boeing 777 aircraft in 1995 [10]: after that, it has been extensively implemented in avionics designs of the US Air Force F-22 commonly integrated processor [11], Airbus 380 [12], Boeing 787 Dreamliner [9], Boeing C130, Gulf-Stream G280, etc.

The importance of temporal correction makes the verification and validation process an essential activity in the avionics development life cycle. In the early design phase,
conduction of a thorough analysis of integrated applications under the design model to evaluate whether or not the design can meet the temporal requirements under IMA architecture has been described in many works [13]–[15]. Furthermore, during the implementation, worst-case execution time (WCET) analysis at the code level produced an estimation of the execution time in order to assure the satisfaction of temporal requirements under the worst-case scenario. The execution order model of a specific CPU was modeled to obtain a static estimation of the execution time of each instruction [16], [17]. Meanwhile, the latency and delay of the IMA network characterized by a bandwidth allocation gap (BAG) were also studied in many works [14], [18], [19]. To ensure that the process execution inter- or inner-IMA partitions can meet their deadlines with given WCET and worst-case communication time (WCCT), another important issue is the schedule algorithm design of the IMA system [20], [21]: a typical two-layer schedule in the IMA system has also been modeled to verify the real-time feature of the system [22], [23], as shown in Fig. 1.

At the top of Fig. 1 is the module-level scheduler schedule of the partitions on the processor for a fixed cyclic length defined by a MAjor Frame (MAF). Each MAF is a periodic schedule where slots are assigned to the partitions based on their period and WCET. Partition slots within MAF are cyclically scheduled nonpreemptively, in which the duration and period are defined by the task 'period and WCET within the partition. Furthermore, as shown in the lower part of Fig. 1, the tasks scheduled within the partition are preemptive, and need to be scheduled within the temporal slot allocated to the partition.

In contrast with the conventional federated architecture, IMA presents the airplane manufacturer with a number of challenges not previously encountered during integration. For example, as implemented on the Boeing 787 Dreamliner, where it is referred to as the Common Core System (CCS), the platform will initially host approximately 70 applications for 20-25 independent suppliers but is capable of hosting over 100 applications [9]. These real-time applications must meet their own temporal requirements while integrating within the context of sharing hardware resources (e.g., processors and communication networks) and software platform (e.g., interface). The system integrator needs to develop a so-called blueprint integration plan to lay out the path to bring each hosted application together and make sure that all resources are properly allocated in order to meet application and functionality requirements and a temporal requirement. Thus, temporally related characteristics of the prospective systems need to be evaluated on the basis of the characteristics of the platform in order to evaluate the complete integration plan.

Many works have been conducted for this issue, and modeling and simulation of the IMA architecture is conducted to evaluate the performance of system [24], [25] at the design level. Considering the detailed design of the architecture, the dynamic schedule algorithm in single and multiprocessors was studied at the partition level [23], and a MILP algorithm was also proposed for formalization of the two-layer scheduler problem and to attempt to produce a possible schedule for the redundancy consideration, in which the exact MILP formulation fails in converging for fairly large problem instances within an acceptable time limit [20]; after that, an algorithm inspired by Game Theory, called the best-response algorithm, was designed for performance improvement [26]. Moreover, an in-depth search algorithm was used to search for the possible configuration of task mapping [15].

Although the model and algorithm previously mentioned both offer numerous contributions to the solution to the task allocation problem from the view of scheduling, little attention has been paid to task mapping with the optimization requirement. With continuous development in avionics, the increasing embedded functions present the integration and configuration with numerous difficulties. Moreover, the evaluation criteria during the integration can be antagonist (such as CPM utilization, communication delay and safety level assurance). In that case, a multiobjective optimization must be introduced for the trade-off. Furthermore, the use of multicore platform and Field-Programmable Gate Array (FPGA) in avionics enables greater flexibility of the IMA system [27], [28] Thus, how to obtain an optimized configuration with respect to a given optimization target within an acceptable timeframe is a problem that must be answered with the full consideration of scalability of both the number of applications and the computing resources.

In this article, the heuristic algorithm was designed to overcome the problems. This algorithm takes both the scheduling constraint and its potential conflicts into account. A classic objective in both research and industry was used as the optimization target. Several comparative studies were conducted on the industry data for the validation of the effectiveness and efficiency of the proposed algorithms under the context of scalability of both the problem domain and computing resources.

The primary contribution of this paper is the presentation of a heuristic solution for task allocation under the optimization requirement during the integration phase. The solution can be more scalable with respect to both task numbers and partition numbers with harmonic or nonharmonic periods and computing resources compared with the previously proposed conventional compositional optimization solution.
Moreover, conduct a comparative study of the two algorithms with respect to effectiveness and efficiency for industrial cases.

The rest of this paper is organized as follows. Section 2 will present the related contributions, terminologies, and research work performed within the domain. The problem definition is elucidated in Section 3. However, the research methodology used in the research study is elaborated in Section 4. Section 5 explains the experimental results and our interpretation, and a discussion on the research analysis will also be argued. In the end, Section 6 summarizes our conclusions and highlights future directions.

II. RELATED WORK

A. TEMPORAL ALLOCATION IN IMA INTEGRATION

Integrated modular avionics (IMA) architectures have promoted a new issue regarding avionics integration [10]. Numerous works have been proposed with respect to different facets of IMA system integration, including incremental integration and certification [29]–[32]. The sharing of hardware platforms makes the architecture integration into a virtual process, which shifts the temporal verification task from the supplier to the integrator [33]. Modeling is a promising method to settle these problems by simulating and validating the virtual integration plan [34], [35]: meanwhile, tools such as the AADL-based tools are designed for the verification of the design [29], and the utilization of timed automata for IMA temporal requirement verification was designed and proven to be helpful. Furthermore, CPM utilization and communication robustness performance criteria are used to quantify the quality of a set of valid temporal allocations [36].

B. SCHEDULE AND RESOURCE ALLOCATION

The precondition of temporal allocation is the schedulability of each task and the whole platform: Al Sheikh et al. formulated a scheduling problem as a mixed-integer linear programming challenge [37]. Furthermore, they addressed this problem by proposing a Game Theory-based algorithm that not only finds a feasible schedule of tasks but also maximizes the relative distances between them [26]. The authors then extended it to support harmonic and near-harmonic periods in the context of IMA [38]. A pseudo-polynomial-time lower-bound algorithm which functions by translating the task assignment problem into the so-called k-cut problem of a graph was proposed in [18] to address the partition allocation problem. Moreover, an in-depth search algorithm is introduced in [15] to allocate the tasks with communication and WCET into partitions.

C. SEARCH-BASED SOFTWARE DESIGN OPTIMIZATION

Harman and Jones [39] proposed search-based software engineering in 2001, reformulated classic software engineering problems as search problems and provided the key ingredients for successful reformulation and evaluation criteria for search-based software engineering. IMA integration can be considered as the possible combination of ARINC653 conformed applications in the whole platform, which has been proved to be an NP-hard problem [40]. The search-based algorithm has also been proven to represent a promising method to settle these kinds of problems. Numerous search algorithms have been proposed, such as classic Linear programming (LP) search, the metaheuristic search algorithm genetic algorithm (GA) [41], simulated annealing (SA) [42], and the Hill Climbing algorithm. The extension of the classic search algorithm and their applications in architecture design optimization with consideration of different quality attributes have been extensively studied, as presented in the systematic literature review in [43].

III. PROBLEM DEFINITION

A. FORMAL DEFINITION OF IMA ARCHITECTURE

From the static view of architecture, typical distributed integrated modular avionics can be illustrated in Fig. 2. The architecture can be decomposed into three parts: Real-time operating system (RTOS), APEX (Application Execution Interface) and Remote devices.

From the top of Fig. 2, RTOS will be deployed into CPM to manage the computing resources and to support the temporal and spatial separation within the CPM. The avionics application represents that the concrete function of the aircraft will be deployed into the RTOS partition (illustrated in Fig. 2 as $T_i$ is deployed in $P_i$). The communication channel within the applications is through the virtual links ($V_j$) and port defined in different partitions, which can be data or command links between Task $T_j$ and end devices through Avionics Full Duplex Switched Ethernet (AFDX). APEX standardized by ARINC653 defines the interface for both RTOS and avionics applications, which facilitates the distributed development of both avionics applications and RTOS. The CPM will connect and control the remote external devices (e.g., sensors ($S_i$) and actuators ($A_i$)) through the remote device control (RDC) component.

A preliminary formal definition of these components represented in Fig. 2 can be described as a series of host tasks $T = T_1, \ldots, T_j$ which will be allocated into a set of partitions $P = P_1, \ldots, P_j$; after that, the partitions...
must be allocated and scheduled on a set of parallel CPMs \( CPM = CPM_1, \ldots, CPM_k \).

**B. TEMPORAL CONSTRAINT IN ARINC653**

As described in the architecture definition, the temporal definitions of tasks, partitions, and CPMs correlate with each other. Formally, from the temporal facet, the task \( i \in Task \) can be described by its worst-case execution time, period and deadline and can then be defined with \( (C_i^F, T_i^P, D_i^F) \). The worst-case execution time represents the maximum temporal requirement that the task needs to finish its required function, the period attribute is the cycle interval of the task to be reactivated, and the deadline is a relative offset from the start of the task, after which the task must be terminated. Similarly, the temporal definition of the partition \( j \in Part \) can be described by the partition execution duration \( C_j^F \) and the period \( T_j^P \), which can be represented in \( (C_j^F, T_j^P) \). However, the attributes of a partition \( (C_j^P, T_j^P) \) are defined by the tasks \( (C_i^F, T_i^P, D_i^F) \) hosted in the partition. The relation between them can be illustrated by Equation (1): \( C_j^P \) is the sum of the WCET tasks ‘while \( T_j^P \) is the greatest common divisor (GCD) of the task ‘period:

\[
C_j^P = \sum_{i=1}^{N} C_i^F \\
T_j^P = GCD(T_i^P), \quad \text{where } T_i \in P_j
\]

Meanwhile, the temporal attribute of \( CPM \) can be described by the MAjor time Frame (MAF), which will cyclically execute the partition scheduled in a static sequence during MAF, where the length of MAF is defined by the least common multiple of the partition host in the CPM, as shown in Equation (3):

\[
MAF_{\text{length}} = \text{LCM}(T_j^P), \quad \text{where } P_j \in CPM_k
\]

**C. SCHEDULABLE CONSTRAINT**

A typical two-layer schedule model in IMA architecture has been extensively investigated by many researchers [20], [21]. One important validation for the correct allocation of the task into partitions is that the temporal requirement has been satisfied. As defined before, the partitioned host in the CPM is scheduled in a fixed and nonpreemptive cyclic MAF, which is pre-defined in the blueprint. The length of MAF is defined as Equation (3). However, in industry, to ensure the maximum utilization of the computing resources and simplify the scheduling algorithm, the periods of partitions in the same CPM are generally harmonic, so that Equation (3) can be rewritten as follows:

\[
\text{MAF}_{\text{length}} = \text{MAX}_{P_j \in CPM_k}(T_j^P)
\]

Considering the task schedule in the IMA, we must define the relation between task and partition and the relation between partition and CPM. The length of the smallest schedule unit in a specific IMA system is defined in Equation (5):

\[
\text{interval}_{\text{length}} = \text{MIN}_{P_j \in CPM_k}(T_j^P)
\]

Moreover, as defined in schedule theory, the necessary but not sufficient condition is that the utilization of the assigned time should be less than the computing time [22] when the system designer must allocate a set of tasks into partitions, to ensure the schedulability of the allocated tasks in the partition, and their relation can be described in Equation (7):

\[
\sum_{i \in \text{interval}} \frac{C_i^F}{\text{interval}_{\text{length}}} < 1
\]

To ensure that all tasks can be schedulable in the partitions, the task assigned in this interval should satisfy the following constraint:

\[
\sum_{i=1}^{n} \frac{C_i^P}{T_i^P} < 1
\]

**D. OPTIMIZATION OBJECT**

The configuration of assigned tasks into partitions can vary with respect to different design considerations. Thus, the efficient use of the CPM computation power can be an important concern during system design. In this section, the evaluation criteria for the task allocation configuration which is frequently used in both industry and research areas [36] are defined as follows.

In general, the allocation needs to achieve a given utilization of computing resources, which can be maximized under the constraint of a certain safe margin. In this paper, the CPU utilization of the CPM is our primary concern: we consider the CPM utilization from a global view, which can be defined as CPM utilization \( Q^{\text{avg}} \). For each CPM, the utilization factor \( Q_i \) is defined as the percentage of time the CPM is executing within a partition, formally:

\[
Q_i = \frac{\sum_{P_j \in CPM_k} C_i^P}{\text{MAF}_k}
\]

where \( CPM_k \) is the allocated computing unit partition, and \( \text{MAF}_k \) is the duration of \( CPM_k \); thus, \( Q^{\text{avg}} \) can be defined as follows:

\[
Q^{\text{avg}} = \frac{\sum_{i=1}^{\lceil m \rceil} Q_i}{m}
\]

where \( m \) is the number of CPMs in the system, and \( Q_i \) and \( Q^{\text{avg}} \) may represent indexes of whether the allocation strategy can ensure sufficient use of CPM.

In practice, thousands of communications and hundreds of CPMs are integrated into an avionics IMA platform. Thus, managing and optimizing the temporal allocation is a very large and complex problem. In one facet, it must satisfy the constraint of the schedulable requirement from the real-time requirement, and in other facet, it must find a sufficiently effective configuration for better CPM utilization under an acceptable limit time. In solving this combinatorial optimization problem, conventional methods such as branch and bound or the MILP algorithm have been proven inefficient at large scale, or even for fairly complex problems, with respect to CPU time or memory space. Thus, whether this complex
problem can be solved by the heuristic algorithms, and how effectively it may be solved, are the questions that must be answered. In that case, the research questions can be defined as follows:

### E. RESEARCH QUESTIONS

**RQ1 (Sanity Check): How do GA and SA perform as compare with random search?** In any attempt at an SBSE formulation of a problem, this is a standard ‘baseline’ question asked. If a proposed formulation does not allow an intelligent computational search technique to convincingly outperform a random search, then there is clearly something wrong with the formulation. This question is thus adopted in SBSE research as a preliminary ‘sanity check ’ [44].

**RQ2 (Effectiveness Comparison): How does GA perform with respect to effectivity compared with SA?** These algorithms may outperform random search and may present different feature in the performance of solution search. Effectiveness is an index of whether the solution can produce a workable configuration with optimized utilization, which is also an important index to judge the validation of the solution. In this research study, effectiveness is an important question that must be answered.

**RQ3 (Efficiency Comparison): How does GA perform with respect to efficiency compared with SA?** Efficiency is another important index for solving the task allocation problem, which will show how rapidly the algorithm can obtain a solution. For that, there must be a balance between the effectivity and efficiency in order to generate a more practicable solution for a certain scenario. In this research study, efficiency is also another important question that must be answered.

**RQ4 (Multicore Acceleration Ratio Comparison): How does GA perform under the multicore scenario compared with SA?** The acceleration ratio is an important indicator of the paralleling ability of both algorithms, which can be used to balance the effectivity and efficiency. Moreover, under the scenario of limited time consumption allowed for obtaining the solution (e.g., online reconfiguration), a solution with a higher acceleration ratio can be easier to apply with multicore platform support.

### IV. METHODOLOGY

The objective of the current research study is to find a possible temporal allocation configuration and achieve a defined optimization objective, maximized CPM utilization, for a given task sequence with the constraint of the schedulable requirement. The methodology used in this research study answers the research question by formulating the temporal allocation task into a search-based format, and then searching for the solution regarding the given target. A heuristic algorithm was chosen to solve the temporal allocation, while the random search was also conducted as a control group to compare the effectiveness and efficiency of the two algorithms. The paralleling heuristic algorithm under multicore context was conducted for the scalability of the proposed problem, while the acceleration ratio of efficiency and effectiveness were also examined and compared with random search. Specifically, the genetic algorithm (GA) and simulated annealing (SA) algorithm are classic algorithms used to find the solution to the allocation scenario which were selected to determine the temporal allocation in the context of IMA integration. Moreover, an experiment in the industrial context was used to prove the effectiveness and efficiency of the chosen algorithm, with each configuration, including 30 epochs of almost 2 hours each, conducted to eliminate the random error. Three typical configurations with the same population and generation as in GA, SA, and Random algorithms were used as the same input of three algorithms to compare the efficiency of the different problems. The main procedure of the methodology is illustrated in Fig. 3:

As shown, the procedure can be divided into three main parts – allocation of the task into partitions, allocation partition into CPM, and evaluation of the fitness – to obtain the final allocation configuration. On the left side of the figure, the first part is the input of the model, which is the temporal attribution of tasks that need to be integrated into the IMA platform with an initial configuration of the application. The second part is the search solution part, which searches for the proper allocation configuration to allocate the previously generated tasks into partitions under the schedulable constraint. The allocation of tasks into partitions has been proven to be an NP-hard problem, for which it is difficult to find the best solution in polynomial time, so that the heuristic algorithm is introduced to settle this problem. The seed solution of the algorithm can be generated by random search. However, in industry, the required time consumption to obtain an allocation configuration is also an important factor that must be taken into consideration (e.g., online reconfiguration). In this research study, a multicore version of the time allocation algorithm was designed to determine whether the algorithm can be accelerated with parallel computing so that it can produce an available solution in a considerably short time. The parallelization of the algorithm is applied to GA and SA so that they can execute paralleling on the multicore platform to reduce the calculated time. The time consumption will be compared with the single-process algorithm to obtain an acceleration ratio. The third part is the simulator, which allocates the previously generated partitions into CPMs. Since different allocations with the same number of CPMs leads to
the same best $Q^{avg}$ for the same tasks, and each partition has two independent attributes, offsets to start and CPMs to which it belongs, the greedy algorithm is applied to acquire the best $Q^{avg}$. In the final part, the ultimate optimization object and the best $Q^{avg}$ will be achieved, which will be used for the effectiveness and efficiency comparison of GA and SA.

A. FORM OF THE SOLUTION AND OPTIMIZED TARGET

Search-based temporal allocation requires the formulation of mapping tasks into the form of search tasks, with corresponding constraints and optimization target. Specifically, the solution of the problem involves determining the mapping relation from tasks to partitions and partitions to CPMs, which contain the temporal configurations of partitions into which a task is allocated, along with its offset in the allocated CPM. In this research study, a $4 \times N$ matrix $M$ is defined as the form of the solution, where $N$ is the number of tasks. The first row of the matrix $M$ represents Task$_1$ to Task$_N$, while the second row contains the partition to which the corresponding task is allocated, which can be defined as $P_1 \cdots P_n$. The third row is the CPM number to which the partition belongs, which is defined as $CPM_1 \cdots CPM_j$. The fourth row is the partition's offset in that CPM. The value of matrix $M$ is $val(M) = Q^{avg}$, which is the optimization target:

$$
M = \begin{bmatrix}
T_1 & T_2 & \cdots & T_n \\
P_{T_1} & P_{T_2} & \cdots & P_{T_n} \\
CPM_{j_1} & CPM_{j_2} & \cdots & CPM_{j_k} \\
offset_1 & offset_2 & \cdots & offset_n
\end{bmatrix}
$$ (10)

The second row will be provided by the search algorithm (TAGA and TASA), while the third row, fourth row and the value of the matrix can be obtained by the simulator. The optimized target can be written as follows:

$$
\text{Max} \{Q^{avg}\} 
$$ (11)

The constraints can be rewritten as follows. The numbers of partitions and CPMS are limited, so that

$$
i < = \text{MaximumPartitionLimit}$$

$$
j < = \text{MaximumCPMLimit}$$

For all CPMs, let $Task_c = \{T_c|m(i, 3) = c\}$, representing all tasks which belong to $CPM_c$, where $1 \leq c \leq j$. The schedulable constraint Equation (12) can then be written as follows:

$$
\forall c, \ 1 \leq c \leq j$$

$$
\forall x(S_1(x) \rightarrow \forall q \forall r(S_2(q) \land S_2(r) \rightarrow \neg(S_3(q, x) \land S_3(r, x))))
$$

where

$$
S_1(x) : x \in \text{interval}_c$$

$$
S_2(x) : T_x \in Task_c$$

$$
S_3(x, y) : T_x^P | (y - M(x, 4))
$$ (12)

The schedulable constraint Equation (14) can then be written as follows:

$$
\sum_{T_i \in Task_c} \frac{C_i^p}{T_i^p} < 1, \ \forall c, \ 1 \leq c \leq j
$$ (13)

The optimization target in Equation (16) can be written as follows:

$$
Q^{avg} = \frac{\sum(C_i^p)/(T_i^p)}{M}
$$ (15)

As illustrated, fewer CPMs leads to a higher $Q^{avg}$ under the same input, while the same amount of CPMs will have the same $Q^{avg}$ under the same input partition list, no matter how the partitions are scheduled.

B. TASK ALLOCATION WITH GENETIC ALGORITHM (TAGA)

The genetic algorithm is applied to allocate the tasks into partitions. In this research study, the task of TAGA is mapping tasks into partitions, which corresponds to the second row of the solution matrix $M$. The search task optimized by GA is formulated as follows: Letting $\overline{X(i)}$ be the population of the $i$th iteration, where $S$, $C$ and $M$ are operators, the algorithm can be written as follows:

$$
\overline{X(i + 1)} = S_2 \cdot M \cdot C \cdot S_1 \cdot \overline{X(i)}, i = 0, 1 \cdots n
$$

where

$S_1$: Select individuals from population

$C$: Crossover from selected individuals

$M$: Mutation

$S_2$: Select best individuals

As is shown in Fig. 4, transition, crossover, mutation, and fitness evaluation are four main processes of TAGA used to produce an optimized offspring for each individual. The evolution of the offspring is generated sequentially, which means that the algorithm will evaluate each individual and optimize that individual towards the optimization target by four processes in each epoch. After each epoch, the algorithm will select the best individual for this iteration and continue until the iteration is finished.

Specifically, the solution was first encoded in the following format, as shown in Fig. 4. A set of $n$ tasks is represented as a string of length $n$, while the partition is labeled in numbers ranging from 1 to $m$. The task here is defined as the gene of the chromosome, and each gene of the chromosome will be assigned to a partition number ranging from 1 to $m$, representing the partition to which the task is allocated.

A single-point crossover is then performed for TAGA species. The crossover point is randomly generated within a range from 0 to $n$, which defines the chromosome for exchange, and the offspring takes a chromosome from both parents, separated by the crossover point, as illustrated in Fig. 6. The first parent is chosen from the population in sequence, and the other parent is determined by Roulette Wheel Selection. The crossover procedure exchanges the task
FIGURE 4. The flowchart of TAGA.

FIGURE 5. Encoding tasks into partitions.

FIGURE 6. Crossover for TAGA: single point crossover.

FIGURE 7. Mutation of TAGA solution.

FIGURE 8. Flowchart of TASA.

allocations of two different individuals. The child1 individual contains the allocation of first half tasks from Parent1, and the rest from Parent2, while the child2 individual contains the allocation of first half tasks from child2, and the rest from Parent1. The entire procedure aims to choose the most excellent genes to create new individuals who might have higher fitness value.

After this, the mutation operation will generate new individuals to avoid falling into local optima. The task is then set to swap the allocations of two tasks chosen randomly from the individual, as shown in Fig. 7. The fitness evaluation of this species is defined as the $Q_{avg}$ of current partition allocation, for which a higher global CPM utilization leads to a higher fitness value.

C. TASK ALLOCATION WITH SIMULATED ANNEALING (TASA)

The simulated annealing algorithm is a classic method to solve allocation optimization, which simulates the annealing process to obtain the best optimization result. As a comparison algorithm, the task of TASA includes mapping between tasks and partitions, which is also the second row of the solution matrix $M$. The solutions are optimized by simulated annealing as follows. Letting $\rightarrow X(i)$ be the population of the $i^{th}$ iteration, where $S$, $C$ and $M$ are operators, the algorithm can be written as:

$$\rightarrow X(i+1) = T \cdot S \cdot \rightarrow X(i), \quad i = 0, 1 \cdot n$$

where

$S$: Select each individual from population

$T$: Transition of selected individual

As is illustrated in Fig. 8: transition, cost evaluation and moving to the neighbor are the three main processes to produce an optimized individual. The neighbor point is chosen in order to attempt different individuals and evaluated by the fitness function. Similarly, similar to GA, these three processes consist of the evaluation of the single individual,
where the sequential processing of all individuals constitutes an epoch. After each epoch, the algorithm will record the best individual and continue until the iteration is finished.

Specifically, the solution is encoded in the following format, as shown in Fig. 10. The allocation of a set of \( n \) tasks to a set of \( m \) partitions is represented as a string of length \( n \). Each position in the string corresponds to a partition number, ranging from 0 to \( m \), representing the partition to which the task is allocated.

A transition is generated by changing the allocation of two randomly chosen tasks, as is shown in Fig. 10. The transition to neighbor procedure exchanges some task allocations of each individual in attempting to find a better solution by searching within its neighbors. The cost of each individual \( E_i \) in the population is considered as the distance between the ideal goal (\( Q_{avg} = 1.0 \)) and the current \( Q_{avg} \) calculated by the simulator, which can be calculated by the formula \( E_i = 1 - Q_{avg} \). The acceptance rate of the neighborhood can be measured by Equation (16):

\[
P = \begin{cases} 
1 & 0 < \Delta E \\
\frac{1}{e^{-\Delta E/T_i}} & 0 > \Delta E 
\end{cases}
\]  

which means that the algorithm always accepts better solutions but may also accept an inferior solution with the probability \( P \). That probability is approximately half at the beginning, but increasingly approaches 0 as iterations proceed. Cooling is performed via the geometric method \( T_{i+1} = \alpha \times T_i \), where \( T_i \) is the current temperature, \( T_{i+1} \) represents the next temperature and \( \alpha \) stands for the temperature decreasing gradient.

D. SIMULATOR

The simulator is designed to allocate partitions into CPMs with a greedy algorithm and is integrated as the fitness evaluator of GA and SA. In addition to the fitness value, the simulator will provide the CPM to which each partition is allocated, along with its offset, which completes the third and fourth rows of the solution matrix. The flowchart of the simulator can be seen in Fig. 10:

As shown, the simulator takes an ordered partition list (OPL) as its input, in which the tasks with long computing time but short period will be allocated first. The simulator will first fetch the first partition from the ordered partition list and allocate it to a new CPM. It will then sequentially scan the ordered input list to find an available partition that can be added to that CPM. This step will be repeated until no more partitions can be added to that CPM. After this, one epoch will finish, and used partitions will be removed from the partition list; the procedure above will be repeated to fill a second partition until the input list is empty. The simulator then evaluates \( Q_{avg} \) by averaging the utilization rate of each core. By simulating the input, the simulator can determine the allocation of partitions into \( M \) CPMs that uses the least CPMs.

To serve as a fitness evaluation, the output of the simulator is deterministic, which means that it will provide the same output if the input for the greedy algorithm scans is the same and will process the input list in the same order.

E. PARALLELING GA AND SA

The paralleling of the algorithm needs to decompose the algorithm into the paralleling part to achieve speed increase under the multicore platform. Specifically, in both algorithms, the evaluation of each individual is the most time-consuming part of the entire process, which needs to be decomposed into independent units. The decomposition of the GA and SA procedures can be seen in Fig. 11 and Fig. 12: As shown, the crossover and mutation procedure in TAGA can be designed as a paralleling, in which the individuals can crossover and mutate on different cores at the same time. The procedure for selection of the best individuals must
evaluate the fitness value of each individual, and so all individual evaluations must be finished before entering the next iteration. However, the parallelization design of TASA is different from that of TAGA, which can be observed in the following figure.

As shown, TASA can parallel all of its procedures, including the selection of the current best individual. This individual can be separately self-optimized without waiting for other individuals. Each individual will achieve the best fitness value, which will then be compared in order to select the best individual after the entire process is complete.

V. EXPERIMENT AND RESULT ANALYSIS

In this section, the synthetic data and configurations are introduced, and the experimental design with result analysis is also presented. Finally, all four research questions will be answered. The purpose of the experiment is to verify the validity of the proposed method in chapter IV. Furthermore, the purpose includes answering the research questions raised in chapter III. A server with 32-core CPUs (4 * E7-4830), 128 GB DDR3 memory, 500 GB HDD, and a Windows Server 2008 R2 system was used to execute the experiments. Approximately 300 hours were required to finish all of the experiments.

A. SYNTHETIC DATA AND EXPERIMENTAL CONFIGURATIONS

A collection of tasks that need to be allocated into partitions and CPMs was used for the experiment. The temporal attributes of the tasks were distinct from those of our previous case study in [25]; however, more simulated tasks in the same pattern were generated, as shown in Table 1. For the most common scenario, in this research study, only the periodic independent task is considered, which is periodically executed in a given period; the relation between tasks is also independent. Thus, each task includes two attributes, the execution time and the period of executing, and the unit of time is normally measured in milliseconds. The period value is generated randomly from the set {20, 30, 40, 60, 100, 120}, while the execution time is a random integer between 1 and 0.2 * period.

The comparison of the effectiveness and efficiency of the algorithms are tested with respect to different configurations. As shown in Table 2, the configuration is divided into two main groups: one is for the single-core platform, and the other for the multicore platform. In the single-core experiment, three configurations were implemented, while four configurations were executed under the multicore platform. Each configuration is applied to three algorithms, GA, SA, and Random, in which the random search algorithm is considered as a control group.

To determine the acceleration ratio of parallelization, a group of another 36 experiments was carried out. Each experiment includes the same 128 tasks to be allocated, as shown in Table 2. As shown, each experiment has a maximum partition number of 100 and a maximum CPM number of 70. However, each experiment owns different population numbers and simulated cores. All configurations will
be executed under both Paralleling GA and Paralleling SA. The acceleration ratio here is simply defined according to the following equation:

$$\text{Acc} \_\text{Ratio} = \frac{\text{Time}_{\text{multicore}}}{\text{Time}_{\text{singlecore}}}$$ (17)

As illustrated in Equation (17), the time consumption for producing an optimized result under the single-core condition is taken as the benchmark, while the ratio is the comparison between time consumption under the multicore condition and the benchmark. Experiments under 1, 2, 4, 8, 16, 20, 24, 28 and 32 cores on the 32-core processor were carried out to determine the trend of acceleration ratio. Moreover, the population numbers of 100, 128, 196 and 256 were used to measure whether the population number influences the acceleration ratio.

Configurations of the experiment I, II, III are repeated 30 times while configurations IV, V, VI, VII are repeated 5 times to reduce the statistical error.

### B. ANALYSIS RESULTS

The objective of this research study is to provide a heuristic solution for task allocation with an optimization objective under the IMA platform. In this section, the results of the comparison experiment are presented, which proved that the heuristic solution is much better than random search with respect to either the single-core or the multicore platform, while the performance of the chosen heuristic algorithm exhibits different features with respect to effectiveness and efficiency. Moreover, at the end of this section, we have encountered all of the research questions that have been presented in the problem definition section.

#### 1) THE SANITY CHECK

The sanity check requires answering whether the problem is suitable for the heuristic algorithm, and furthermore, by what percentage the algorithm outperformed random search. After the execution of configurations I, II, III for 30 times each, the static results can be observed in Table 3 and Table 4.

As illustrated in Table 3, the least, average and highest numbers of CPM are used to represent the allocation results. The results show that both GA and SA can provide a better solution than random search in all three configurations (less CPM is used), even in the configuration III, where the least number of CPM is equal. However, GA exhibits the best result with respect to CPM utilization for all three levels, as shown by the numbers labeled in bold. Finally, the utilization of the CPM can be calculated from Equation (17).

The utilization rate is shown in Table 4, where three static indexes (Best, Average, Worst) are used to represent the performance of each configuration and the highest rate in each static index is labeled in bold. The results show that both SA and GA can provide a better solution than a random search in all three configurations. Moreover, SA achieved the six highest utilization rates in three configurations, while GA produced the two highest utilization rates in the best static column. The results can be observed more directly in Fig. 13: As is shown, in each configuration, SA and GA perform much better than random search. This is especially true in configuration I and configuration II, which do not contain much space for the allocation. Even in configuration III, where the best solutions are much more easily found, the static indexes from SA and GA are still superior to random search.

#### 2) EFFECTIVENESS AND EFFICIENCY COMPARISON

During the experiment, the best utilization rate in each iteration and its time consumption were recorded for further analysis. The utilization rates of the solutions produced by three algorithms during the optimization process with configurations I, II, III are shown in Fig. 14. The average value of the best utilization rate found for the same iteration during 30 repeated experiments is used to eliminate the random error. It can be observed that the utilization rates of workable task allocation found by GA and SA rise rapidly during the first 50 iterations and keep ascending in the following iterations. However, the utilization of random search grows much more slowly and stops improving after approximately 200 iterations. Specifically, GA provides a better solution in the first 153 iterations in configuration I and the first 194 iterations in configuration II, after which the utilization of SA overtakes GA. However, GA is always better than SA in configuration III.

In addition to the effectiveness comparison, the efficiency will determine the time to produce an optimized result. The results obtained during the experiment can be observed in Fig. 15. As illustrated, GA is more efficient than SA under a given lower optimization target (e.g., 0.950 in configuration I and configuration II). After that point, SA can continuously

### TABLE 3. Statistics of the CPM used for task allocation.

| CPM | Configuration I | Configuration II | Configuration III |
|-----|-----------------|------------------|-------------------|
|     | Least | Average | Highest | Least | Average | Highest | Least | Average | Highest |
| GA  | 18    | 26.30   | 33       | 18    | 26.56   | 34       | 15    | 16.13   | 27       |
| SA  | 18    | 22.63   | 27       | 18    | 22.63   | 27       | 15    | 17.87   | 22       |
| R   | 19    | 23.04   | 30       | 19    | 23.00   | 31       | 15    | 18.38   | 25       |

### TABLE 4. Statistics of utilization rate.

| Q\textsuperscript{52} | Configuration I | Configuration II | Configuration III |
|------------------------|-----------------|------------------|-------------------|
|                         | Best | Average | Worst | Best | Average | Worst | Best | Average | Worst |
| GA                     | 0.992 | 0.975 | 0.956 | 0.984 | 0.973 | 0.957 | 0.991 | 0.987 | 0.980 |
| SA                     | 0.994 | 0.986 | 0.979 | 0.991 | 0.987 | 0.980 | 0.994 | 0.985 | 0.982 |
| R                      | 0.949 | 0.931 | 0.916 | 0.946 | 0.930 | 0.917 | 0.997 | 0.984 | 0.980 |

#### FIGURE 13. Utilization rate comparison.
optimize the result, while GA increases slowly, which means that SA will be more efficient for a given higher target.

However, GA is always better in configuration III and can produce an optimized result in a shorter timeframe. Therefore, GA is more suitable when the calculation time is limited, or when the target utilization rate is not especially high, especially in the configuration I and configuration II.

To compare the solutions provided by GA and SA, a Wilcoxon signed-rank test [45] is applied. This test is an alternative nonparametric test that is widely used in validation.

The results of the test are presented in Table 5. The p-values reported in Table 5 conform with the box-plots, indicating that SA is superior to GA, while GA is better than Random with respect to configurations I and II. GA outperforms SA with respect to configuration III. In other words, SA performs better than GA with respect to configurations I and II, while GA is superior with respect to configuration III.

### TABLE 5. The results of the Wilcoxon signed-rank test.

| Configuration   | P-value    |
|-----------------|------------|
| I               | III        |
| GA vs SA        | 2.19E-07   | 1.59E-3    |
| GA vs Random    | 2.87E-11   | 2.87E-11   |
| SA vs Random    | 2.87E-11   | 2.87E-11   |

3) MULTICORE ACCELERATION RATIO COMPARISON

As mentioned in section IV, paralleling versions of TAGA and TASA are designed to reduce the time consumption involved in searching the optimization result. The average running times of SA and GA are illustrated in Table 6 for the numbers of cores ranging from 1 to 32.

As illustrated, the average acceleration ratios for the two algorithms over different numbers of cores decreases obviously. The lowest time consumption is for the SA version under the 100 population on 32 cores, and the rate of decrease of SA is much higher than that of GA.
TABLE 6. Time consumption (minutes) of paralleling searching.

| pop  | 1  | 2  | 4  | 8  | 16 | 20 | 24 | 28 | 32 |
|------|----|----|----|----|----|----|----|----|----|
| SA   |    |    |    |    |    |    |    |    |    |
| 100  | 17 | 8.9| 4.5| 2.4| 1.4| 1.2| 1.1| 1.1| 1.1|
| 128  | 23 | 11 | 5.8| 3  | 1.7| 1.5| 1.4| 1.3| 1.2|
| 192  | 34 | 17 | 8.8| 4.5| 2.5| 2.2| 2  | 1.8| 1.7|
| 256  | 46 | 22 | 12 | 5.9| 3.2| 2.9| 2.5| 2.4| 2.2|
| GA   |    |    |    |    |    |    |    |    |    |
| 100  | 56 | 28 | 15 | 8.6| 6.2| 6.6| 7  | 7.5| 8.4|
| 128  | 90 | 46 | 23 | 18 | 9.4| 9.2| 9.4| 10 | 10 |
| 192  | 204| 101| 55 | 30 | 20 | 20 | 20 | 21 | 24 |
| 256  | 367| 187| 97 | 53 | 37 | 38 | 37 | 37 | 39 |

FIGURE 16. Comparison of acceleration ratios.

According to the acceleration ratio defined in Equation (17), the final acceleration ratio can be observed in Fig. 16. As demonstrated by the light gray lines in Fig. 16, the acceleration ratio of paralleling SA monotonically increases upon the addition of more computational resources. However, the black lines show that adding more computational resources cannot benefit GA further when the number of cores is approximately 16 to 24. The peak acceleration ratio of GA occurs when the numbers of cores are 16, 20, 192, and 256, while the population numbers are set to 100, 128, 192, and 256, respectively.

More interestingly, it can be observed that the acceleration ratio of SA is much higher than that of GA, especially after GA reaches its peak. After inspecting the solutions generated during the process, the reason preventing GA from benefiting from parallelism might be attributed to the reproduction process of GA. The reproduction process of GA generates a large number of invalid solutions that are time-consuming to validate or repair. Moreover, the communication overhead for sorting and comparing individuals in this population becomes the key factor in running time when parallelism within a population increases to a certain level. Therefore, adding cores is effective when paralleling SA, but will be less useful when paralleling GA, especially when there are more than approximately 16 cores.

4) ANSWERS TO RESEARCH QUESTIONS

The objective of this study was accompanied by four research questions. After the formulation of the industry data into the heuristic algorithm and conduction of the designed experiments, we are now in a state of responding to the answers of research questions.

RQ1 (Sanity Check): How do GA and SA perform compared with random search? After formulating the temporal allocation problem into the heuristic search optimization, both effectiveness and efficiency of the solution can be compared with random search. As shown in Fig. 11, and as observed for the experimental data, both GA and SA can provide better solutions than random search. Moreover, both GA and SA perform better than random search, especially in situations where good solutions are more difficult to find, as in configurations I and II.

RQ2 (Effectiveness Comparison): How does GA perform with respect to effectiveness compared with SA? Effectiveness mainly focuses on the result of the optimization target, which is the average utilization of the CPM in our research study. Both algorithms perform well, but SA is more effective in the CPM limited situation, while GA performs much better when time is limited, as is shown in Fig. 12 and Table 3. Moreover, with the Wilcoxon signed-rank test, the solutions found by SA are higher than those of GA.

RQ3 (Efficiency Comparison): How does GA perform with respect to efficiency compared with SA? Efficiency pertains to the time required to obtain an optimized solution. In our experiment, GA performs more efficiently when there is a strict time limit or a relatively minimally optimized target. However, SA is more efficient in other situations, such as when there is no time limit or when the solution needs to be as good as possible. These results show that both algorithms have applicable areas, which may depend on the specific scenario.

RQ4 (Multicore Acceleration Ratio Comparison): How does GA perform under the multicore scenario compared with SA? As shown in the experiment, both SA and GA can be accelerated to a relatively large ratio under the multicore scenario. However, due to the algorithm structure of the TAGA, adding more computational resources can only benefit GA to a certain degree, after which the defects of GA overtake the benefits gained from parallelism. Moreover, the acceleration ratio of SA is monotonically increasing in our experiments as more computational resources are added for the independent structure of the paralleling decomposition of individual evaluation.

VI. CONCLUSION

Ample work has been conducted on the modeling of temporal relations under ARINC653 [13], [36], [46] and attempting to find possible temporal allocation configurations for a given task and platform [15], [47]. Furthermore, the solution of the temporal allocation must consider the WCET of the task, health monitoring strategy, the dependency relation, communication delay, safety assurance level and even the requirements of the certification agency. Therefore, for the deterministic requirement of the avionics system design, little attention has been paid to the solving of the task allocation under optimization target with a probabilistic algorithm.
However, with the development of the ARINC653-based platform, increasing applications are decomposed into tasks, so that the allocation and the optimization will eventually become important tasks during the virtual integration of the IMA system, and even within the online reconfiguration module. In this scenario, a workable allocation configuration or the configuration with an optimized target, or even under considerably small delay, form the basis of the decision support or even the reconfiguration module.

In this study, the mapping of the independent task allocation under ARINC653 conformed architecture is solved by the heuristic algorithm. Two classic heuristic allocation algorithms, GA and SA, are chosen for the experiment, and the parallel version is also designed for the low delay requirement of possible avionic applications. By combining industry data and simulated data, experiments are carried out to discover the effectiveness and efficiencies of the chosen heuristic algorithm.

The results show that both GA and SA are much better than random search: SA is more effective in the CPM limited situation, while GA performs much better when the time is limited. Moreover, GA performs more efficiently when there is a strict time limit or a relatively low optimization target. In the scenario of acceleration by multicore, SA performed much better than GA, which may indicate a potential choice for further application.

The algorithms presented in this paper have answered the problem but also raise some new questions that need to be discussed in the future. There are two primary aspects that may represent the future sector. First, the algorithms provide solutions within reasonable time, they are still too slow to meet the needs of online reconfiguration. Thus, parallel computing methods, such as GPU acceleration, multithreading or FPGA, can be applied to the current algorithm. However, parallel computing methods may lead to some potential safety issues such as exchanging information, which also must be taken into consideration. Second, the tasks in industry are more complex and include more attributes such as network latency, multicore processors, limited maximum CPM usage and associated tasks, as mentioned previously. This indicates that the algorithms must be modified by inclusion of additional elements and constraints in order to satisfy more critical requirements, which may represent the direction of further research.

VII. ACKNOWLEDGMENT
We dedicate this work to Prof. C. Liu, who unfortunately passed away just before the paper was submitted for publication. Prof. Liu served in an essential role in the research described herein, and he is greatly missed by all of us.

REFERENCES

[1] S. Asif and P. Webb, “Software system integration-middleware—An overview,” Int. J. Comput. Appl., vol. 121, no. 5, pp. 27–29, 2015.

[2] S. Hamdan and S. Alatrnouni, “A quality framework for software continuous integration,” Procedia Manuf., vol. 3, pp. 2019–2025, 2015.

[3] M. Dias, V. Uquillas-Gomez, D. Cassou, and S. Ducasse, “Software integration questions: A quantitative survey,” Tech. Rep. 01093496, Laboratoire d’Informatique Fondamentale de Lille, Lille, France, 2014.

[4] D. A. M. Montana, D. F. C. Rodriguez, D. I. C. Rey, and G. Ramos, “Hardware and software integration as a realistic SCADA environment to test protective relay control,” IEEE Trans. Ind. Appl., vol. 54, no. 2, pp. 1208–1217, Mar. 2018.

[5] S. Mirheidari, A. Fallahi, D. Zhang, and K. Kuppam, “AUTOSAR model-based software component integration of supplier software,” SAE Int. J. Commercial Vehicles, vol. 8, no. 2, pp. 544–548, Sep. 2015.

[6] H. Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications. Springer, 2011.

[7] GulfStream. Gulfstream 280/aviation application. Accessed: Jan. 26, 2018. [Online]. Available: https://www.rockwellcollins.com

[8] A. Tsourdos, “Flight control systems: Practical issues in design and implementation,” Proc. Inst. Mech. Eng., vol. 214, no. 5, p. 323, 2000.

[9] C. B. Watkins and R. Walter, “Transitioning from federated avionics architectures to integrated modular avionics,” in Proc. IEEE/AAIA 26th Digit. Avionics Syst. Conf., Oct. 2007, p. 2.

[10] R. Garside and F. Pighetti, “Integrating modular avionics: A new role emerges,” IEEE Aerosp. Electron. Syst. Mag., vol. 24, no. 3, pp. 31–34, Mar. 2009.

[11] C. Spitzer, U. Ferrell, and T. Ferrell, Digital Avionics Handbook. Boca Raton, FL, USA: CRC Press, 2014.

[12] H. Butz, “Open integrated modular avionic (IMA): State of the art and future development road map at Airbus deutschland,” Signal, vol. 10, p. 1000, 2010.

[13] N. Badache, K. Jaffres-Rusner, J.-L. Scharbarg, and C. Fraboul, “End-to-end delay analysis in an integrated modular avionics architecture,” in Proc. IEEE 18th Conf. Emerg. Technol. Factory Autom. (ETFa), Sep. 2013, pp. 1–4.

[14] M. Li, J. Ermont, F. Boniol, and C. Pagetti, “Latency and freshness analysis on IMA systems,” in Proc. ETFa, Sep. 2011, pp. 1–8.

[15] E. Deroche, J.-L. Scharbarg, and C. Fraboul, “Mapping real-time communicating tasks on a distributed IMA architecture,” in Proc. IEEE 21st Int. Conf. Emerging technol. Factory Autom. (ETFa), Sep. 2016, pp. 1–8.

[16] J. Souyris, E. L. Pavec, G. Himbert, G. Borios, V. Jègu, and R. Heckmann, Computing the Worst Case Execution Time of an Avionics Program by Abstract Interpretation (OpenAccess Series in Informatics (OASIcs)), vol. 1, Dagstuhl, Germany: Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.

[17] P. Lokuciejewski and P. Marwedel, Worst-Case Execution Time Aware Compilation Techniques for Real-Time Systems. Berlin, Germany: Springer, 2010.

[18] C. Ekelin and J. Jonsson, “A lower-bound algorithm for minimizing network communication in real-time systems,” in Proc. Int. Conf. Parallel Process., Jun. 2003, pp. 343–351.

[19] H. Charara, J.-L. Scharbarg, J. Ermont, and C. Fraboul, “Methods for bounding end-to-end delays on an AFDX network,” in Proc. 18th Euromicro Conf. Real-Time Syst. (ECRTS), Aug. 2006, p. 10.

[20] A. A. Sheikh, O. Brun, and P.-E. Hladik, “Partition scheduling on an ima platform with strict periodicity and communication delays,” in Proc. 18th Int. Conf. real-time Neww. Syst., 2010, pp. 179–188.

[21] L. Sha, T. Abdelzaher, K.-E. Årzen, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, “Real-time scheduling theory: A historical perspective,” Real-Time Syst., vol. 28, nos. 2–3, pp. 101–155, Nov. 2004.

[22] Y.-H. Lee, D. Kim, M. Younis, and J. Zhou, “Scheduling tool and algorithm for integrated modular avionics systems,” in Proc. 19th DASC. 19th Digit. Avionics Syst. Conf., vol. 1, Nov. 2004, pp. 1–8.

[23] J.-E. Kim, M.-K. Yoon, S. Im, R. Bradford, and L. Sha, “Multi-IMA partition scheduling with synchronized solo-partitions for multi-core avionics systems,” Dept. Comput. Sci., IDEAR, Chicago, IL, USA, Tech. Rep. handle/2142/35280, 2012.

[24] C. Fraboul and F. Martin, “Modeling advanced modular avionics architectures for early real-time performance analysis,” in Proc. 7th Euromicro Workshop Parallel Distrib. Process. (PDP), 1999, p. 181.

[25] K. Zhang, J. Wu, C. Liu, S. S. Ali, and J. Ren, “Behavior modeling on ARINC653 to support the temporal verification of conformed application design,” IEEE Access, vol. 7, pp. 23852–23863, 2019.

[26] A. Al Sheikh, O. Brun, P. E. Hladik, and B. J. Prabhhu, “A best-response algorithm for multiprocessor periodic scheduling,” in Proc. 23rd Euromicro Conf. Real-Time Syst., Jul. 2011, pp. 225–237.
K. Zhang et al.: Independent Temporal Integration of ARINC653 Conformed Architecture — Search Based Solution

[27] T. Gaska, C. Watkin, and Y. Chen, “Integrated modular avionics—past, present, and future,” IEEE Aeros. Electron. Syst. Mag., vol. 30, no. 9, pp. 12–23, Sep. 2015.

[28] B. Sutterfield, J. A. Hoschette, and P. Anton, “Future integrated modular avionics for jet fighter mission computers,” in Proc. IEEE/AIAA 27th Digit. Avionics Syst. Conf., Oct. 2008, p. 1.

[29] A. Khosrashilov, D. Alibitskiy, I. Koverninskii, M. Olshanskii, A. Petenko, and A. Ugennko, “AADL-based toolset for IMA system design and integration,” SAE Int. J. Aerosp., vol. 5, no. 2, pp. 294–299, Oct. 2012.

[30] A. Wilson and T. Preyssler, “Incremental certification and integrated modular avionics,” IEEE Aeros. Electron. Syst. Mag., vol. 24, no. 11, pp. 10–15, Nov. 2009.

[31] G. Bartley and B. Lingberg, “Certification concerns of integrated modular avionics (IMA) systems,” in Proc. IEEE/AIAA 27th Digit. Avionics Syst. Conf., Oct. 2008, p. 1.

[32] R. L. Eveleens, “Integrated modular avionics development guidance and certification considerations,” Mission Syst. Eng., vol. 2, pp. 1120–1132, 2006.

[33] P. H. Feiler, J. Hansson, D. D. Niz, and L. Wrage, “System architecture virtual integration: An industrial case study,” Software Eng. Inst., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-2006-TN-011, 2009.

[34] J. Delange, L. Pautet, A. Plantec, M. Kerboeuf, F. Singhoff, and F. Kordon, “Validate, simulate, and implement arinc653 systems using the aadl,” ACM SIGAda Ada Lett., vol. 29, no. 3, pp. 31–44, 2009.

[35] T. Robati, A. Ghebi, A. El Kouhen, and J. Mullins, “Design and simulation of distributed IMA architectures using TTEthernet: A model-driven approach,” J. Ambient Intell. Humanized Comput., vol. 8, no. 3, pp. 345–355, Jun. 2017.

[36] N. Badache, K. Jaffres-Runser, J.-L. Scharbarg, and C. Fraboul, “Managing temporal allocation in integrated modular avionics,” in Proc. IEEE Emerg. Technol. Factory Autom. (ETFA), Sep. 2014, pp. 1–8.

[37] A. Al-Sheikh, O. Brun, and P.-E. Hladik, “Decision support for task mapping on IMA architecture,” in Proc. 3rd Junior Researcher Workshop Real-Time Comput. (JRWRCT), 2009, pp. 31–34.

[38] Q. Zhang, S. Wang, and B. Liu, “Approach for integrated modular avionics reconfiguration modelling and reliability analysis based on AADL,” IET Softw., vol. 10, no. 1, pp. 18–25, Feb. 2016.

[39] M. Harman and B. F. Jones, “Search-based software engineering,” Inf. Softw. Tech., vol. 43, no. 14, pp. 833–839, 2001.

[40] J. Korst, E. Aarts, and J. K. Lenstra, “Scheduling periodic tasks,” INFORMS J. Comput., vol. 8, no. 4, pp. 428–435, 1996.

[41] J. H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1992.

[42] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, “Equation of state calculations by fast computing machines,” J. Chem. Phys., vol. 21, no. 6, pp. 1087–1092, Jun. 1953.

[43] A. Aleli, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, “Software architecture optimization methods: A systematic literature review,” IEEE Trans. Softw. Eng., vol. 39, no. 5, pp. 658–683, May 2013.

[44] M. Harman, S. A. Mansouri, and Y. Zhang, “Search-based software engineering: Trends, techniques and applications,” ACM Computing Surv. (CSUR), vol. 45, no. 1, p. 11, 2012.

[45] A. Shukla, H. M. Pandey, and D. Mehrotra, “Comparative review of selection techniques in genetic algorithm,” in Proc. Int. Conf. Futureistic Trends Comput. Anal. Knowl. Manage. (ABLAZE), Feb. 2015, pp. 515–519.

[46] A. Al Sheikh, O. Brun, P.-E. Hladik, and B. J. Prabhu, “Strictly periodic scheduling in IMA-based architectures,” Real-Time Syst., vol. 48, no. 4, pp. 359–386, Jul. 2012.

[47] D. Tamas-Selicean and P. Pop, “Task mapping and partition allocation for mixed-criticality real-time systems,” in Proc. IEEE 17th Pacific Rim Int. Symp. Dependable Comput., Dec. 2011, pp. 282–283.

KUI ZHANG received the master’s degree in information management and information systems from the Beijing Institute of Technology, Beijing, China, in 2010. He is currently pursuing the Ph.D. degree with the Software Engineering Institute (SEI), Beihang University. His research interests include model-driven engineering, model-based real-time analysis, airworthiness certification, model-based safety analysis, and general model-based software engineering.

JUNCHEN LIU is currently pursuing the bachelo’s degree with the School of Computer Science and Engineering, Beihang University, Beijing, China. His research interests are in the areas of search-based software engineering, big data analysis, software engineering, and heuristic algorithms.

JIAN REN received the M.Sc. degree from the Queen Mary University of London, the M.Sc. degree from King’s College London, and the Ph.D. degree in computer science from University College London. He is currently an Assistant Professor with the School of Computer Science, Beihang University, Beijing. His research interests include search-based software engineering, software project planning and management, requirements engineering, and evolutionary computation.

JINGHUI HU received the bachelor’s degrees in computer science and technology, mathematics and applied mathematics and the master’s degree in computer technology from Beihang University. He is currently an Assistant Engineer with the AVIC Manufacturing Technology Institute. His research interests include intelligent manufacturing technology, search-based software engineering, and machine learning.

CHAO LIU received the M.S. and Ph.D. degrees in computer software and theory from Beihang University. He was a Professor with the School of Computer Science and Engineering, Beihang University. His research interests include software quality engineering, software testing, model-driven software development, and software process improvement. During the preceding decade, he primarily focused on the modeling and verification of safety-critical software and systems, including safety requirement modeling and analysis, evidence-based software safety analysis and evaluation, software safety and reliability analysis based on the software development process, and model-driven software testing.