computers

Article

An Evolutionary Approach for the Hierarchical Scheduling of Safety- and Security-Critical Multicore Architectures

Brandon Woolley [1,*], Susan Mengel [2] and Atila Ertas [3]

1 Space and Airborne Systems Raytheon Company 2501 W. University, McKinney, TX 75071, USA
2 Department of Computer Science, Texas Tech University 7th and Boston Lubbock, Lubbock, TX 79409, USA; susan.mengel@ttu.edu
3 Department of Mechanical Engineering Texas Tech University 7th and Boston Lubbock, Lubbock, TX 79409, USA; atila.ertas@ttu.edu
* Correspondence: brandon.woolley@ttu.edu

Received: 7 July 2020; Accepted: 31 August 2020; Published: 3 September 2020

Abstract: The aerospace and defense industry is facing an end-of-life production issue with legacy embedded uniprocessor systems. Most, if not all, embedded processor manufacturers have already moved towards system-on-a-chip multicore architectures. Current scheduling arrangements do not consider schedules related to safety and security. The methods are also inefficient because they arbitrarily assign larger-than-necessary windows of execution. This research creates a hierarchical scheduling framework as a model for real-time multicore systems to integrate the scheduling for safe and secure systems. This provides a more efficient approach which automates the migration of embedded systems' real-time software tasks to multicore architectures. A novel genetic algorithm with a unique objective function and encoding scheme was created and compared to classical bin-packing algorithms. The simulation results show the genetic algorithm had 1.8–2.5 times less error (a 56–71% difference), outperforming its counterparts in uniformity in utilization. This research provides an efficient, automated method for commercial, private and defense industries to use a genetic algorithm to create a feasible two-level hierarchical schedule for real-time embedded multicore systems that address safety and security constraints.

Keywords: hierarchical scheduling; multicore; genetic algorithm; safety and security

1. Introduction

Next-generation aerospace and defense (A&D) applications will have to contend with both stringent safety and security requirements; hence, software written over the last 30 years for the Department of Defense programs faces a challenging migration problem. In addition, migrating legacy software from older single-core processing environments must be rearchitected to target modern multicore processing architectures.

A compounding issue that must be addressed when moving to modern architecture frameworks are safety and security requirements [1]. Parkinson defines, in [2], safety as where the system must not harm the world and security as where the world must not harm the system. These requirements constrict the program's ability to select any commercial operating system. Historically, programs with safety of flight requirements used what is called a partitioned operating system [3,4]. A partitioned operating system separates (or partitions) task execution in both time and space. The operating system uses a static partition schedule to guarantee task execution separation in time. For separation in space, the partitioned operating system separates the task execution in memory. The partitioned operating system enforces separation so that if one task crashes, it does not affect the rest of the system. Programs that have information assurance requirements use certified operating systems known as separation kernels. Fundamentally, partitioned operating systems and separation kernels provide the same time and space separation.

To address the issues stated above, separation kernel vendors are leveraging the multilevel instruction mechanisms (hypervisor, supervisor, and guest) enabled by multicore hardware and are now offering virtualization support. The implication of virtualization is that the legacy uniprocessor operating system can be placed within the separation kernel's operating system as a guest operating system. Gaska, in [5], highlight the benefits of virtualization that allow for interoperability, legacy software migration, and board consolidation. Placing guest operating systems within separation kernels on multicore systems creates a more effective use for multicore architectures, which is echoed by Kotha and Nasser in [6] and Main in [7]. Another consideration for multicore architectures is the mode in which the processor will operate: asymmetric multiprocessing (AMP) versus symmetric multiprocessing (SMP). AMP mode, from a task and real-time operating system (RTOS) perspective, arranges the cores into separate processing modules where SMP provides a monolithic "one-processor" view. Given the nature of aerospace and defense legacy uniprocessor designs, AMP is the mode better suited to satisfy the safety and security requirements discussed in [2], because legacy software is not written to take advantage of the parallel nature of SMP. Using AMP gives rise to yet another issue highlighted by Leroux and Craig [8], that management of shared resources such as memory, Input/Output devices, and interrupt handling falls to the engineer, not the operating system. Kleidermacher, of Green Hills Software, Inc., a real-time embedded operating systems vendor, points out in [9] the need to migrate to multicore processing and offers the approach using virtualization. This approach is inadequate because it neglects the up-front work associated with migration, as well as the sustainment and refinement phases that follow.

Current research on hierarchal scheduling has neglected legacy scheduling architectural arrangements. For example, rate monotonic scheduling, which is prevalent in legacy scheduling systems, is rarely considered. This research is unique because it combines the two most prevalent scheduling algorithms within A&D safety and security systems: rate monotonic scheduling and static cyclic scheduling algorithms. The purpose of the genetic algorithm is to provide an automated means to find a feasible two-level hierarchical schedule. The use of an evolutionary optimization method provides enhanced determinism through uniformity in utilization. This paper highlights the value of this research over traditional bin-packing methods by providing greater determinism. This type of determinism is better because it evenly distributes tasks among resources (cores). Bin-packing algorithms tend to overload frontend resources and underload backend resources, leading to considerable variation in core utilization. This research is unique because the genetic algorithm (GA) and its objective function provide more even distribution/utilization across the cores, resulting in better determinism, which is more efficient than current bin-packing methods.

This research defines the architectural framework and creates an evolutionary optimization method for embedded real-time multicore two-level hierarchical scheduling that provides enhanced determinism through uniformity in utilization. The goal of this research is to enable legacy real-time embedded systems to be effectively managed during the migration and analysis of their real-time software tasks to multicore system-on-a-chip architectures. This research provides a method of optimization that naturally finds a feasible two-level hierarchical schedule with a uniform utilization configuration. First, literature relevant to hierarchical scheduling approaches for embedded real-time software tasks that include safety and security requirements is examined and evaluated. Techniques used to analyze two-level hierarchical scheduling are investigated, which include the evaluation of crucial timing parameters for the specific combination of schedules at each level. The scheduling algorithms, techniques, and critical timing parameters are then used to construct the task model necessary for analysis. Secondly, an evolutionary approach for scheduling software tasks, specifically a GA, is created, which involves the creation of a novel chromosomal encoding method and the construction of an objective function that aids in measuring the fitness of potential candidate solutions. Finally, this research leverages and builds upon traditional bin-packing heuristics used to schedule tasks and integrates these algorithms into a validation framework. A task schedule modelling and simulation methodology is built to facilitate the validation of the newly created two-level hierarchical schedule. The next section of the paper provides a review of the relevant literature that builds the foundation of this research.

2. Related Works

2.1. Hierarchical Scheduling

For real-time systems, there are two types of scheduling approaches: static or fixed priority and dynamic priority schedulers [10]. Static schedulers are more desirable for embedded airborne applications because they provide enhanced determinism. In hard real-time systems, schedulability is analyzed in terms of response times and verified to ensure determinism. Schedulers are evaluated based on their ability to meet task deadlines. Evaluation implies that a valid (optimal) schedule is one where all task deadlines are met. Brandenburg, in [11], clarifies the term optimality in the context of schedulability: a scheduler is said to be optimal if it is able to find a valid schedule for a given set task if one exists. Finding a valid schedule is an important distinction when considering fixed or static scheduling policies as the designer must create an optimal scheduler a priori.

The two schedulers considered in this research are a Fixed-Priority Preemptive Scheduler and a Static Partitioned Scheduler. From their seminal work on schedulability analysis, Liu and Layland [12] set the groundwork that spawned rate monotonic analysis (RMA) and rate monotonic scheduling (RMS) for hard real-time priority preemptive scheduling systems. Dhall and Liu [13] first researched partition scheduling for real-time multiprocessor systems. They developed a simplified heuristic-based approach for task-to-processor allocation that statically partitioned the tasks among the available processors. The issues surrounding the problem, first introduced by [13], of scheduling task sets to processing resources is a classic bin-packing problem. Bin-packing is a method of allocating items to resources (bins). For scheduling, it is synonymous to assigning tasks to a resource (a core). Bin-packing is known to be NP-Hard. Garey and Johnson, in [14], proved that it is not feasible to find an optimal schedule due to the large amount of computational overhead. The concept of NP-Hard is crucial because genetic algorithms are known to perform well in finding solutions to NP-Hard problems [15]. Since the introduction of multicore, there has been a scramble to comprehend the added complexity of scheduling.

Lipari [16] writes that hierarchical scheduling is the formation of a number of schedulers into a tree structure. Also, the scheduling hierarchy can have an arbitrary number of levels, nodes, and children, where the nodes can be a task or another scheduler. Hierarchical scheduling approaches as a method of analysis of schedulability have been applied to non-real-time and real-time systems in the past [17,18]. Different methodologies for designing hierarchical two-level schedules have been proposed in [19].

Recent advancements in embedded real-time multicore system-on-a-chip designs that allow for multiple levels of scheduling have inspired renewed attention to the field. The first known application and proposal of the hierarchical scheduling framework (HSF) using a real-time operating system is in [20]. Asberg applied a hierarchical scheduling framework to the real-time operating system VxWorks. The HSF was created to exert greater control over the tasks and the resources they share. Later, Asberg et al. published their initial results, in [21,22], and minor updates to the HSF with a goal of a future research to extend HSF to multicore architectures, found in [23].

Cullmann et al., in [24], discussed how embedded system timing analysis can become infeasible when moving to multicore architectures. The authors recognized the issues surrounding multicore and shared resources (memory hierarchy, etc.). Finally, they provided some design principles for moving single-core embedded systems to multicore ones that improved predictability. The goal of the design principles is to architect the system in such a way to allow better bounding of the worst-case scenario for improved analysis. These design principles are difficult to apply to legacy systems given the author's use of abstraction. A means of modeling the complexities of existing systems is still needed for the required level of determinism for the A&D industry. There is no tolerance for the error an abstraction may produce. For example, a streaming service may experience glitching without consequence; however, if a missile does not fire when the trigger is pulled, lives are at stake.

Hilbrich and Goltz [25] provided a model-based approach to simplify multicore scheduling complexities. Their approach is to model the software's resource utilization on a multicore system and schedule it accordingly. They developed a tool, named "PRECISION PRO", that accepts a task set and their parameters as input to generate a static partition schedule that maps the tasks to partitions and then cores. The authors claim enhanced resource utilization, flexibility, and automated verification. Their approach only considered a first-level scheduler and primarily dealt with new developments targeting multicore systems, but can be extended to evaluate legacy migrations.

vSlicer is a scheduling approach for virtual machines that Xu et al. described in [26] for the IT server-level market. The purpose is to address the issue of I/O-bound latency-sensitive virtual machines. Their approach is used to reduce other latency, jitter, and performance issues. Although this scheduling algorithm is meant to address these issues in the IT server market, it has applicability to the real-time embedded market given that it has to contend with virtual operating systems.

Presented in [27] with successive improvements in [28,29], Nemati et al. actually implemented an approach to solve the partition and intrapartition scheduling issue with resource sharing considerations. Their approach is the most detailed and practical implementation written to date. One drawback is that it did not account for a guest operating system, but the framework provided is scalable and extendable.

Kwon et al. [30] presented a method for a hypervisor to schedule virtual machines more efficiently on asymmetric multicore systems. Their method allowed for increased portability to multicore systems by obscuring the machine-dependent asymmetry of the hardware. This method assumes that the same hypervisor manages all the cores within the multicore architecture. Though not an ideal solution, Kwon et al. illustrated yet another potential solution to the hierarchical scheduling problem.

Masrur et al. [31] recognized the apparent two-level scheduling issues that arise from using virtual machines and guest operating systems in embedded multicore systems. They studied the effect of virtual machines used in embedded systems with a hierarchical two-level schedule. They also pointed out that traditional hierarchical schedulability conditions do not apply to this embedded virtual machine scenario. The proposed methods by Masrur et al. have merit and will be considered further for this research.

2.2. Scheduling with Genetic Algorithms

Genetic algorithms come from the field of study called evolutionary computation and are biomimetically inspired by evolution [32]. Genetic algorithms are a class of evolutionary optimization methods [33–35]. Primarily used as an optimization technique, genetic algorithms offer effective methods to search stochastically, in parallel, the solution space. Jensen and Chryssolouris [36,37] note that creating schedules is an NP-Hard optimization problem and determine that genetic algorithms are far superior at finding efficient schedules. Wall [38] points out that genetic algorithms are well suited for the complex combinatorial nature that most scheduling problems exhibit. Genetic algorithms are used in a variety of different scheduling systems; for example, job shop scheduling systems [36–41], real-time systems [42–45], distributed and grid computing systems [15,46,47], and multiprocessor systems [48–51].

Wall [38] also points out when tackling a complex problem, designing the right heuristic is difficult. Two of the most important aspects for the development of an effective GA are the objective function and the parameterization [32]. The objective function should be applicable to all the schedules developed by the algorithm. The parameterization, on the other hand, identifies the unique set of parameters that need to be updated for each specific case.

Montana et al. [52] note that genetic algorithms are well suited for the complexity that scheduling presents due to their ability to search large spaces. They developed a scheduling approach using genetic algorithms for field service engineers for the military land move problem (a common convoy scheduling problem for moving from fort to seaport). More specifically, they provide an analysis of how to create a chromosomal representation for different types of scheduling problems. This analysis has broad applications and is applicable to this research's aim of creating a chromosome.

Oh et al. [44] used a genetic algorithm to allocate real-time tasks to a multiprocessor system with the goal of minimizing the number of processors used. Using a multiobjective genetic algorithm, the authors showed that their approach outperformed other algorithms on 178 randomly generated task graphs used in their experiments. The authors also developed a method for determining the optimal scheduling of real-time tasks to a homogeneous multiprocessor system.

Kaur et al. [51] illustrated the complexities of multiprocessor task scheduling in high-performance computing environments. The authors used a genetic algorithm for static nonpreemptive scheduling problems in an online fashion. They compared the results to a uniprocessor control and two other multiprocessor scheduling algorithms. Their approach produced a better scheduling solution for online or active scheduling of tasks to processors and outperformed the other algorithms in both time to schedule and efficiency. This approach illustrates promising results for this proposed research.

3. Hierarchical Scheduling Framework

Preliminary attempts have modelled systems and their complexities, but the inclusion of guest operating systems via virtual machines on multicore processors are an order of magnitude more complex than current methods can handle. Figure 1 illustrates just how complex these types of systems have become. Each core contains multiple partitions and multiple guest operating systems with tasks. The management of such an organization of elements is a complex combinatorial problem.

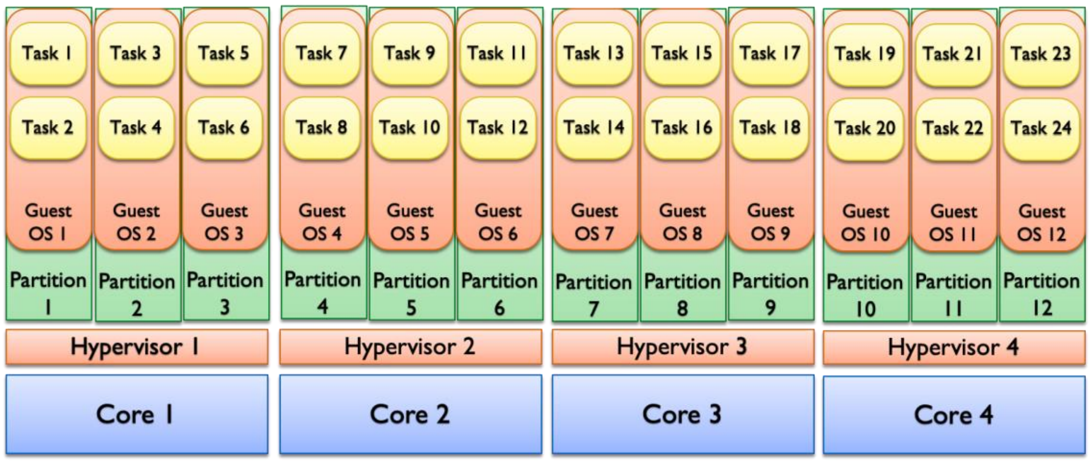

Figure 1. Example multicore asymmetric multiprocessing (AMP) mode with guest operating system (OS) virtualization.

Figure 1 is a relatively simple four-core implementation, but eight, twelve, and sixty-four core architectures exist, increasing system complexity exponentially. Predictability of the scheduling effects of virtualization utilizing guest operating systems (two-level scheduling) becomes more difficult to model, manage, and analyze. The motivation of this research is to assemble the pieces of the various disciplines to provide a unified methodology for automatically generating a two-level hierarchical multicore schedule.

System Model Definition

Typical applications measure performance in a variety of different ways utilizing many different scheduling algorithms. For legacy real-time systems, one of the most common real-time operating systems (RTOS) is VxWorks. Other, less well-known, RTOSs used in safety-critical applications are partitioned operating systems (POS). To combine these two operating systems, a system model is constructed.

A task, τ_i, is defined as the basic unit of execution and is characterized by the worst-case execution time (WCET), defined as C_i; a relative deadline D_i; and a period of T_i. A sporadic task will produce a series of jobs with interarrival times separated by T_i time units. The cost of each job is C_i and the relative deadline is D_i, which occurs after the job's arrival time. The tasks are sporadic with implicit deadlines where $D_i = T_i$ for all τ_i (also known as Liu and Layland [12] task systems).

The parameters for schedulability are specific to the task/partition system at each level. At the first level is the static cyclic schedule (SCS), and at the second level is the rate monotonic schedule (RMS). From the task's perspective, the RMS is at the first level and then the SCS is at the second level. The hierarchical system model is composed of two levels. The first level consists of the partition model, and the second level consists of the task model. Tasks are the basic unit of execution of the overall system model. For this research, each task is considered independent and static with no other resource requirements. Tasks have a cost, period, and deadline (note: for rate monotonic scheduling, $D = T$ is assumed). Partitions are also composed of cost and period in a similar manner as the task. The total partition cost is calculated by the summation of all task costs within a partition. The period is simply the smallest task period that occurs within a partition.

Definition 1. *For n number of tasks, the tasks τ_i are organized into task sets $\Gamma = \{\tau_i, \tau_{i+1}, \ldots, \tau_{i+n}\}$ that are then assigned to the specific partition π_i.*

Definition 2. *Partitions are gathered into partition sets, $\Pi = \{\pi_i, \pi_{i+1}, \ldots, \pi_{i+n}\}$, that are in the end assigned to the required number of cores, m_i.*

A given task set is scheduled using the second-level rate monotonic scheduling algorithm. Rate monotonic scheduling is calculated using an exact result method called the worst-case response time (WCRT). WCRT is the task delay between the release time and the end time. WCRT analysis is computed with the following expression [11]:

$$r_i = C_i + \sum_j \left\lceil \frac{r_i}{T_i} \right\rceil C_j \tag{1}$$

where j is the remaining tasks that have a higher period (T) than task i. To determine taskset schedule feasibility, we use WCRT and iteratively compute [11]:

$$w_i^{n+1} = C_i + \sum_j \left\lceil \frac{w_i^n}{T_i} \right\rceil C_j \tag{2}$$

The pseudocode for the RMA schedule feasibility is given in Algorithm 1.

Algorithm 1 Pseudocode of RMA Schedule Feasibility

```
1. Sort taskset $\Gamma$ in priority order (lowest period is highest priority)
2. Start with $w_i^0 = C_i$
3. Loop over each task and test for convergence using Equation (2) (ergo $w_i^1, w_i^2, w_i^3, \ldots\ w_i^k$)
4. If $w_i^k > T_i$, no task response time can be computed for task $i$. Tasks will not converge Schedule is unfeasible.
   Break and quit
5. Else, if $w_i^k = w_i^{k-1}$, $w_i^k$ is the response time for task $i$. Tasks will converge. Schedule is feasible. Break
   and continue for all tasks in taskset
```

A partition set is scheduled using the first-level static cyclic scheduling algorithm. For a given partition set Π, with T_i as its period and C_i as its WCET, we apply a multiconstraint-based algorithm from [53]. The pseudocode for the SCS schedule feasibility is given in Algorithm 2. For SCS, the major frame is computed first. The minor frame requires the satisfaction of a couple of constraints. First, we find the largest C_i to use it as the lower bounds of all possible minor frames. Secondly, we find the smallest T_i to use it as the upper bounds of all possible minor frames. This will minimize the number of possible context switches. Next, we minimize the overall table size (number of entries) by finding all possible minor frame values using the modulus of the major frame by the minor frame. Finally, for all partitions, we test every minor frame for feasibility and return a pass/fail based on success or failure.

Algorithm 2 Pseudocode of SCS Schedule Feasibility

```
1.  Sort partition set $\Pi$ in priority order (lowest period is highest priority)
2.  Compute the major frame $M$ using lowest common multiple (l cm) of partition periods
3.  $M = \text{l cm}(T_i)\ \forall\ \pi_i$ in $\Pi$
4.  Find the upper and lower bounds of all partitions such that:
5.  $\forall\ \pi_i$ in $\Pi$ compute (i = Min($T_i$); i < max($C_i$); i++)
6.    if ($M$ % i == 0) $F_i \leftarrow$ i
7.  $\forall\ F_i$ in $F$ find $\pi_i$
8.    $\forall$ partition's $T_i$ in $\Pi$ compute
9.      if (2*$F_i$ − gcd($F_i$, $T_i$) <= $T_i$)
10.       return pass
11.     else return fail
```

The scheduling arrangement seeks to allocate tasks to partitions and partitions to cores in an efficient manner while distributing the total utilization across all cores, as illustrated in Figure 2.

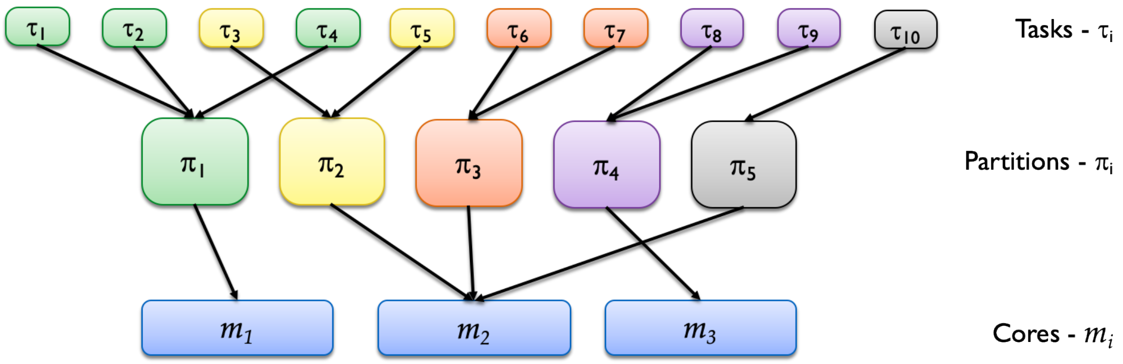

Figure 2. Schematic of the allocation of tasks to partitions to cores.

The tasks are assigned to partitions which have guest operating systems that employ a fixed-priority preemptive rate monotonic scheduler. The partitions are assigned to cores that employ a static cyclic scheduler (also known as the partition scheduler).

Finally, consider a similar illustration with the combined schedules. Figure 3 illustrates a virtualized guest operating system running within each partition schedule. Figure 3 illustrates an example of the complexity of migrating to a multicore system. From the system model in Figure 2, the reader can see how the partitions π_2, π_3, and π_4 combine their minor frames to construct the major frame in Figure 3. The major frame is repeated over and over with each corresponding minor frame within. It is important to note that each partition contains an internal taskset that must be scheduled to execute within a guest operating system.

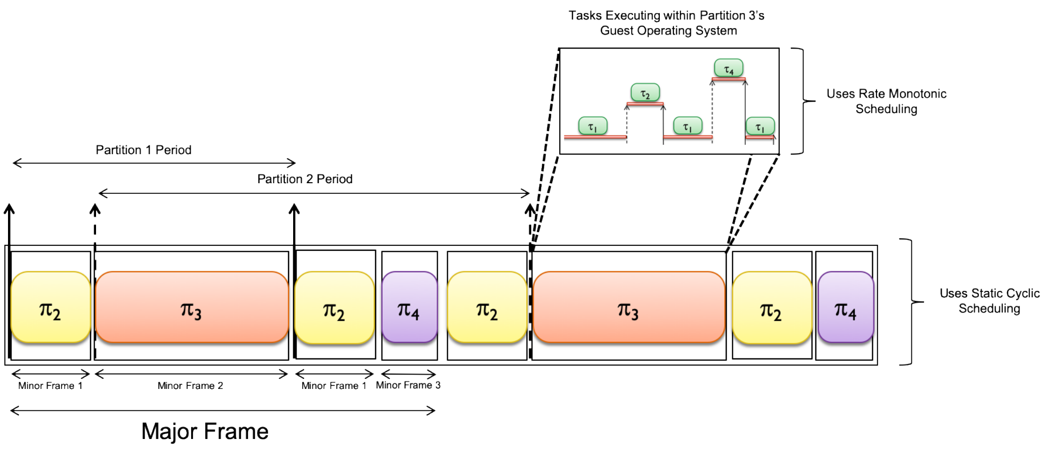

Figure 3. A logically combined two-level schedule.

Figures 2 and 3 show the hierarchical relationship of the scheduling problem we solve. A benefit highlighted above is that two different legacy uniprocessor tasksets are migrated and consolidated onto a single core, but combining both schedulers in a hierarchical fashion presents a complex problem. Special analysis must be performed to determine task schedulability to address this problem. One major issue is that the guest operating system has a fixed-priority preemptive scheduler that orders the tasks based on their frequency of execution, where the static partition scheduler blindly moves through its major/minor frame schedule. If the fixed-priority preemptive scheduler's frequency of execution is incongruent with the static partition schedule of the major frame, then deadlines and response times will be missed. A final factor to contemplate is that only one core has been considered so far. Multicore architectures come in a variety of arrangements that can vary from 4, 8, 12, to 24 cores. As the number of cores increase, an exponential expansion is created that drives up the scheduling complexity: imagine the complexity illustrated in Figure 3 multiplied by 24 or even 64 times in a cascading fashion. The creation and management of 64 different two-level hierarchical schedules becomes a daunting task. Therefore, to facilitate migration to a multicore system, the translating model has to comprehend the two-level hierarchical scheduling relationships.

A hierarchical scheduling framework (HSF) is constructed for the integration of the two types of schedules identified in this research. At the first level is a static major and minor frame partition scheduler, and at the second level is a task-based fixed-priority preemptive scheduler. Both scheduling algorithms are well-researched, well-understood, and are most commonly used in the aerospace and defense industry. The purpose of this framework is to provide a method of determining the essential schedulability thresholds for a two-level hierarchical schedule. The fixed-priority preemptive scheduler is relatively easy to analyze using rate monotonic analysis (RMA). The partition scheduler is made more complicated to analyze due to the addition of the concept of the guest operating system and its internal scheduler. The HSF and its schedulers are used to construct the task model necessary for analysis.

4. Genetic Algorithm Scheduling Approach

The application of a genetic algorithm (GA) to the newly constructed two-level hierarchical scheduling framework involves the adaptation of two essential GA functions: an encoding/decoding method and the creation of an objective function. The encoding/decoding method crafts a chromosomal makeup of the essential parameters that determine schedulability in real-time embedded operating systems used in legacy systems. Particular attention is given to encoding approaches for scheduling that include the consideration of multicore architectures to aid the identification of the critical attributes and parameters that are applicable to this research. Another important aspect of the application of GAs is the construction of an objective function that measures the fitness of potential candidate solutions. The objective function needs to accurately and consistently capture the properties of the multiple objectives integral to the two-level hierarchical scheduling problem presented in this research. The purpose is to create a scoring method that considers the appropriate fitness for the problem at hand.

This research focuses on how previous genetic algorithm encoding schemes are generated for other real-time and non-real-time scheduling applications. Particular attention is given to encoding approaches for scheduling that include the consideration to real-time applications. The investigation identifies the essential parameters that are used to determine schedulability. Using these parameters for schedulability, the genetic algorithm's chromosome that formulates a task's schedulability is created.

Once the parameterization portion of the genetic algorithm construction is complete, the formation of an objective function that measures the fitness of potential candidate solutions is generated. The objective function needs to capture the properties of the multiple objectives integral to the two-level hierarchical scheduling problem. To aid in this construction, the information on two-level hierarchical schedulability is used to evaluate the schedule for acceptance (i.e., feasibility).

The method must take into account fitness and schedule utilization for the problem at hand with the objective of enhancing the selection method with an expanded solution space. The purpose is to create a selection process that strikes the right balance between the search for fitter candidate solutions (exploitation) and at the same time promoting beneficial diversity in the population (exploration). To balance exploitation with exploration, the ranking mechanism within the GA is modified to take into account the aforementioned considerations.

4.1. Chromosomal Encoding Method

The genetic algorithm requires an encoding method that allows for both a task-to-partition assignment as well as a partition-to-core assignment at the same time. The possible encoding methods include 1D, 2D, and 3D chromosomal representations. As with many solutions, dimensionality adds complexity. 2D and 3D chromosomal representations can facilitate an adequate mapping for core, partition, and task assignments; however, they would separate the individual alleles from each other. This separation will cause multiple evolving independent chromosomes that would have to be consolidated into a single solution. Our solution requires an integrated chromosomal encoding method because the schedulability of tasks and partitions must be tested at each epoch. Hence, a 2×1D method is chosen because it lessens the dimensionality burden while allowing the solution space for cores, partitions, and tasks to be searched simultaneously. Kaur and Singh [51] provide an encoding example that is adapted to our research. We leverage the idea of chromosomal encoding that provides a processor-to-task assignment and expand it to account for task-to-partition-to-core assignment.

Figure 4 provides a visual representation of the 2×1D encoding method used to represent our hierarchical scheduling framework's task-to-partition-to-core assignments. From left to right and top to bottom, the initial chromosome is randomly generated and evolves into the final chromosomal encoding representation that is uniformly schedulable.

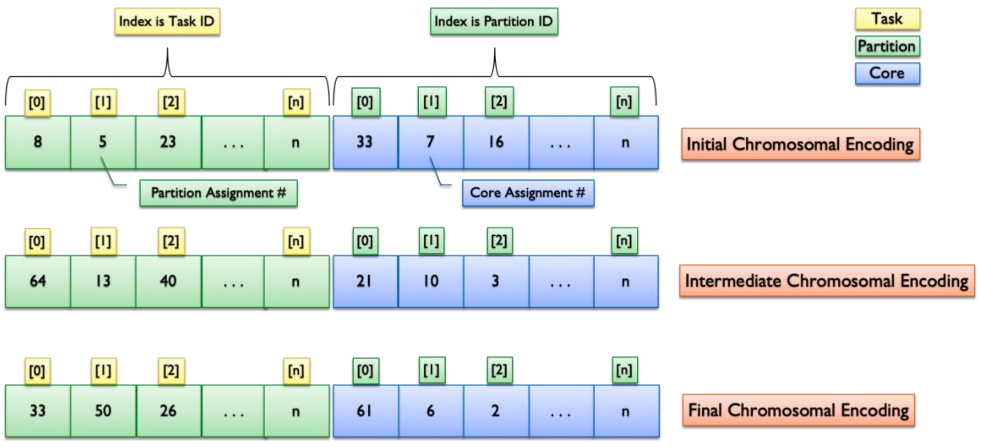

Figure 4. The chromosomal encoding method.

4.2. The Objective Function

For this research, the GA needs to select the best task arrangement that provides the target processor utilization while satisfying the two-level schedule's feasibility tests. The best candidate must have the highest score. Genetic algorithms use an objective function to score potential candidate solutions. This research leverages the objective function, a critical element with the GA framework, to satisfy our desired scheduling approach. By normalizing the core utilization as the objective, the GA creates a uniform schedule.

A multicriterion scoring model is used to satisfy both levels of schedulability as well as to test for total processor utilization. The goals of the scoring model are to measure all criteria on a similar scale and assign weights to each criterion based on its importance. This scoring methodology is used as part of the genetic algorithm's objective function that provides feedback on each chromosome's population rank.

Each chromosome is tested for schedulability at both levels, and a total core utilization factor is calculated. From [38], we consider a simple piecewise chromosome score that considers two constraints and one objective for the entire genetic algorithm objective function score. Each schedule's feasibility, f_c, given in Equation (3) is a constraint to the consideration of the core utilization objective score, U_{obj}, given in Equation (4).

The rate monotonic, RM_S, and static cyclic, SC_S, schedule feasibility scores are counted on a per core basis. The partition and task schedule feasibility constraint methodology counts the number of nonfeasible schedules for a particular chromosome. The equation used provides a method by which a larger number generates a lower overall score. As the individual RM_S and SC_S constraint scores tend to zero, the combined score approaches 1. The final constraint score is calculated by:

$$f_c = \frac{RM_s + SC_s}{2} \tag{3}$$

If the constraint score is less than 1, the chromosome score equals the constraint score. Also, if the constraint score is ever greater than 1, then the core utilization (the objective score) is added to the overall chromosome score.

Core utilization objective, U_{obj}, is only counted if it is less than or equal to 100%. If so, it is summed into a total chromosomal core utilization value. This value is then divided by the number of cores allocated for this chromosome to obtain an average core utilization value. The core utilization value is normalized and used as the objective score for the chromosome. The objective score is then added to the constraint score and divided by 2 to provide the final chromosome score that is returned back to the genetic algorithm for consideration for the next generation. The core utilization is calculated by:

$$U_{obj} = \sum_{i=1}^{n} U(m_i),\ if\ U(m_i) \le 1.0 \tag{4}$$

The overall combined objective function score, S_{obj}, given in Equation (5), for a chromosome is calculated using the following expression:

$$S_{obj} = \frac{U_{obj} + f_c}{2} \tag{5}$$

The reason for the uniform utilization across cores is due to the constraint score satisfaction required before core utilization becomes part of the objective. This means that only 'feasibly' scheduled chromosomes will be considered for their core utilization, thereby training the GA to spread the tasks and partitions amongst the available cores.

5. Evaluation Methodology

In order to validate the two-level hierarchical schedules, a standard comparison methodology proposed in [54] was adopted. The validation effort used software tools and frameworks used for modelling and simulating schedules for real-time embedded systems with the purpose of developing a method to model, simulate, analyze, and validate the generated two-level hierarchical schedule. The first subtask investigates and builds upon an existing genetic algorithm software library to create the feasible two-level hierarchical schedule. The next subtask synthesizes the results of the genetic algorithm into a form that can be used as an input to the selected modelling and simulation environment. The identified attributes and parameters were used for encoding a two-level hierarchical schedule. A modelling framework was created to accommodate the multicore two-level scheduling environment discussed earlier. Finally, the framework was used to test and validate the newly formed two-level hierarchical schedules.

Figure 5 shows the experimental platform and detailed configuration information used in our simulations.

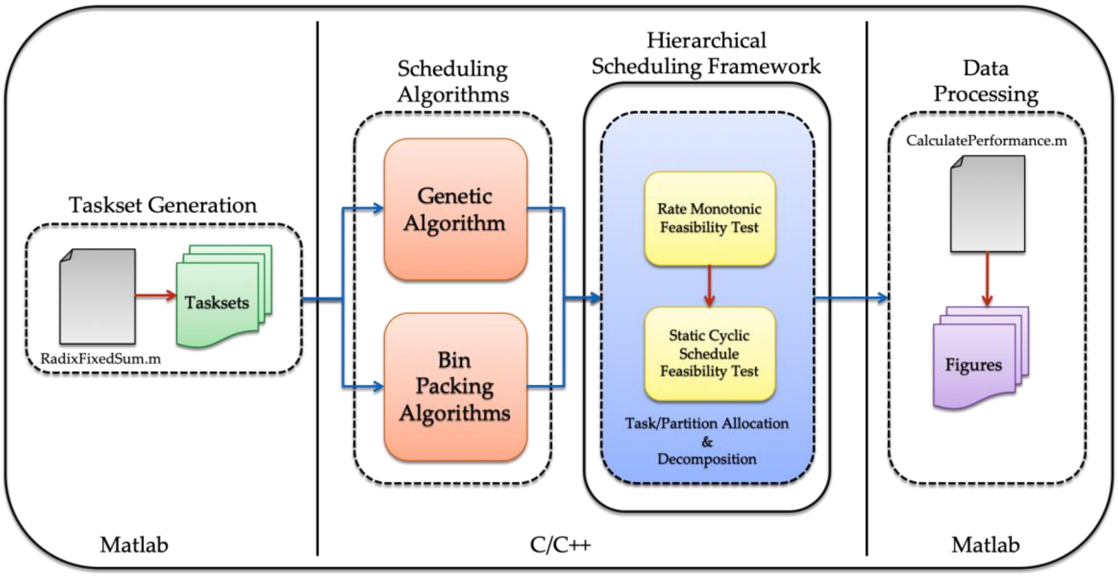

Figure 5. The experimental platform and detailed configuration information.

From left to right, taskset generation creates tasksets and is generated using Matlab, the scheduling algorithms and hierarchical scheduling framework are run using C/C++, and finally, data processing calculates performance using Matlab. The programs were run on a PC with an Intel 3.2 GHz 8-Core Intel Xeon W processor and 32 GB 2666 MHz DDR4 running on the MacOS platform. More detailed information on relevant methods are provided in the following subsections.

5.1. Data Generation

The experiments are performed on randomly generated task sets. Several techniques for creating task data sets for analysis have been proposed in the literature [55,56]. The tasks are generated using methods based on the UUnifast-Discard task set generation algorithm in [56] and the radixfixedsum task set generation algorithm which is similar to that found in [57]. Each algorithm creates a set of tasks with an associated period and cost. Task data is generated into groupings by core (2, 4, 6, 8 or cores). A core consists of a number of tasks where total utilization is equal to a fixed value starting at 80% and incrementally increasing by 5% up to 100%. Task numbers per core are stepped in increments of 5 to 20. The tasks' periods are varied from 10 to 100, 10 to 200, 10 to 500, and 10 to 1000. Each core, task, and utilization range are created for the various period ranges. A task's periods have a greater effect on schedulability than cost. Therefore, task periods are randomly selected within these ranges. For example, a data run of 5×2 at 0.85 is a core number of 2 with each containing 5 tasks with utilization of 85%. In total, the number of tasks created is 20,000.

5.2. Heuristic Comparision

Bin-packing is a well-studied field for efficient item allocation that provides an industry benchmark of schedule performance [54,58–60]. The heuristics are first fit, next fit, worst fit, best fit, and increasing or decreasing sorted versions. Each heuristic is defined as follows:

- First fit (FF): Of the available bins, allocate the item into the bin it fits into first. If no available bin exits, create a new one.
- Next fit (NF): Allocate the item into the current bin. If it does not fit, open a new bin. Never cycle through already opened bins.
- Worst fit (WF): Search the available bins and allocate the next item into the emptiest bin. If two bins tie, select the first available emptiest bin. If the item does not fit into any available bin, open a new bin.
- Best fit (BF): Of the available bins, search through and place the next item into that bin which will leave the least room left over after the item is placed in the bin. If the item does not fit in any bin, open a new bin.

For the variations of increasing and decreasing item sizes, each heuristic is employed on the sorted list of items. FF, BF, and their sorted counterparts first fit decreasing (FFD) and best fit decreasing (BFD) provide the near optimal and optimal solutions for sorting [58] and are used for comparison to the genetic algorithm.

5.3. Performance Measures

Two measures of performance (MOP) are used to provide quantitative measures:

- Mean squared error (MSE): the average squared loss
- Total core count (TCC): total number of cores consumed

MSE is widely used in machine learning to quantify the difference between the forecasted and actual core utilization. MSE is also known as the average squared loss per example over the entire dataset. To calculate MSE, we sum up all of the squared losses for each of the scheduling attempts and then divide by the total number of attempts. Total core count is used to provide quantitative comparison amongst the scheduling algorithms.

6. Results and Discussion

6.1. Experimental Arrangement

The experiments were composed of the development of several computer programs that aid the creation of sample data that is fed into the hierarchical scheduling framework. Some key assumptions and preconditions were made to simplify the simulation itself. The details of the current implementation are included below.

Here are some basic assumptions related to the simulation program for this research:

- Tasks are considered periodic and independent;
- Task periods and execution cost are known a priori;
- The guest operating systems use fixed-priority preemptive rate monotonic scheduling;
- The partitions scheduler uses static cyclic scheduling.

The method of evaluation used to determine performance of the hierarchical scheduling framework and genetic algorithm creates a comparative heuristic-based simulation on top of the same hierarchical scheduling framework.

6.2. Simulation Results

Twenty runs of a thousand tasks required normalization of the results presented. The results presented were organized by the period range examined. Within each period range, the measures of performance were graphed, including the number of cores consumed for task schedulability and the average error from each algorithm. After analyzing the data, several relevant patterns emerged. The genetic algorithm achieved a similar number of cores to the bin-packing algorithms; however, the GA outperformed the bin-packing algorithms in average loss.

Illustrated in Figure 6, the average number of cores used was six for all algorithms, where the average loss varied by algorithm. Generally speaking, the ordered bin-packing algorithms tended to outperform their unsorted counterpart. For the period range 10–100, the sorted bin-packing algorithms (BFD and FFD) had a larger error rate when compared to their unsorted counterparts (FF and BF). The GA had 3.6% to 4.3% less error (a 65–69% difference) when compared to the bin-packing algorithms, which means that the GA outperformed the bin-packing heuristics in its ability to match the expected scheduling uniformity across the same number of cores.

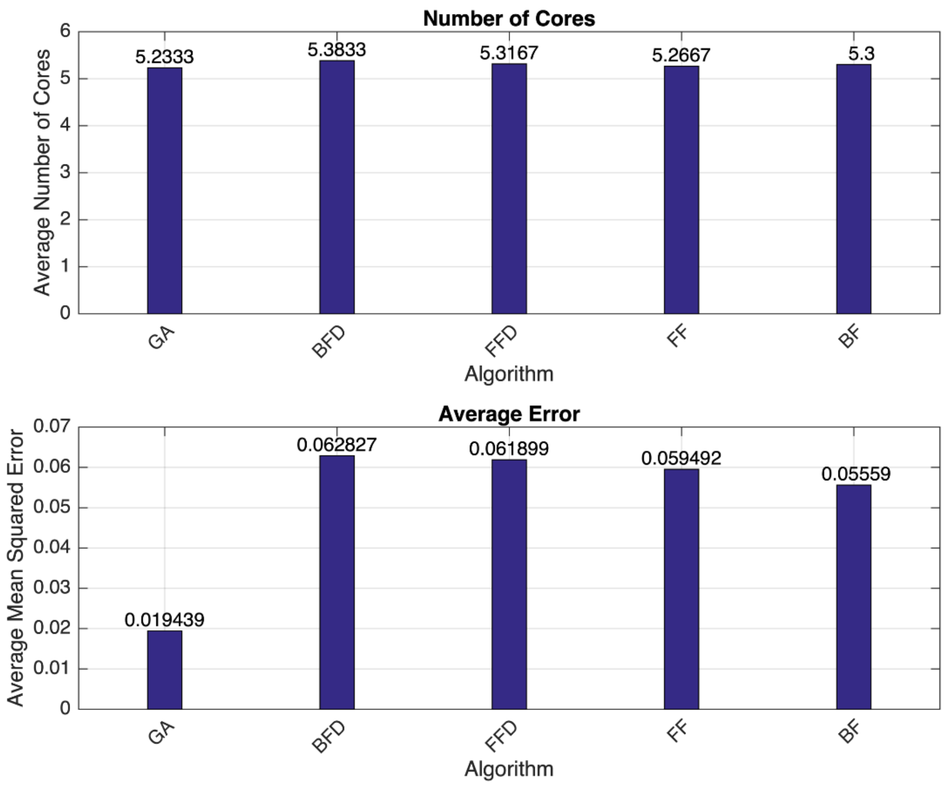

Figure 6. Results for the period range 10–100. GA: genetic algorithm; BFD: best fit decreasing; FFD: first fit decreasing; FF: first fit; BF: best fit.

In Figure 7, the average number of cores consumed was six for all algorithms, where the average loss varied by algorithm. Generally speaking, the ordered bin-packing algorithms tended to outperform their unsorted counterparts (BF and FF). For the period range 10–200, the sorted bin-packing algorithms (BFD and FFD) had a similar error rate when compared to their unsorted counterparts. The GA had 3.5% to 4.0% less error (a 56–60% difference) when compared to the bin-packing algorithms, which means that the GA outperformed the bin-packing heuristics in its ability to more closely match the expected scheduling uniformity across the same number of cores.

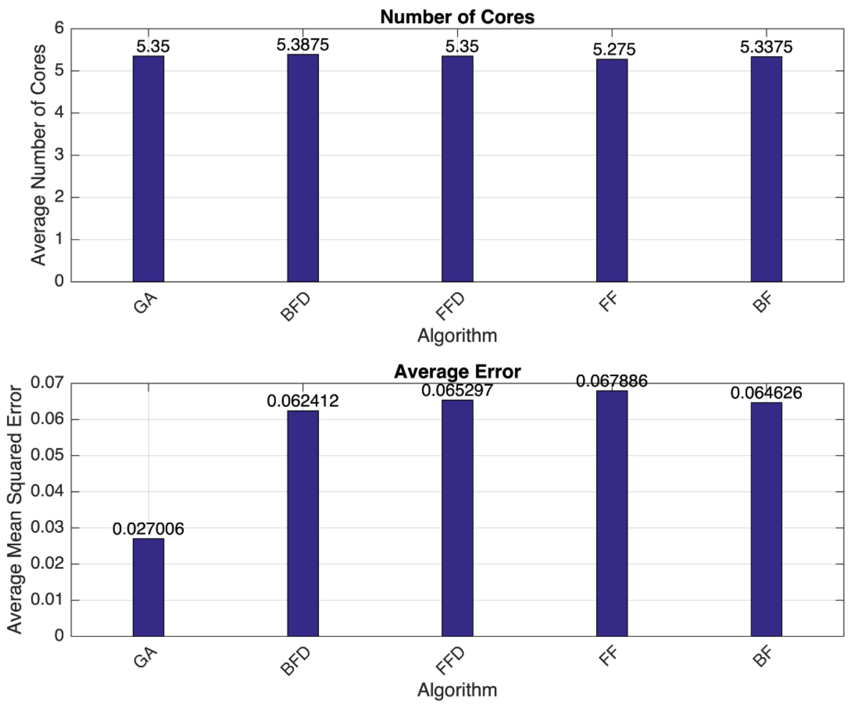

Figure 7. Results for the period range 10–200.

As illustrated in Figure 8, the average number of cores used was six, with rounding, for all algorithms, where the average loss varied by algorithm. Generally speaking, the ordered bin-packing algorithms tended to outperform their unsorted counterparts (BF and FF). For the period range 10–500, the sorted bin-packing algorithms (BFD and FFD) had larger error rates when compared to their unsorted counterparts. The GA had 4.0% to 5.4% less error (a 60–67.5% difference) when compared to the bin-packing algorithms, which means that the GA outperformed the bin-packing heuristics in its ability to more closely match the expected scheduling uniformity across the same number of cores.

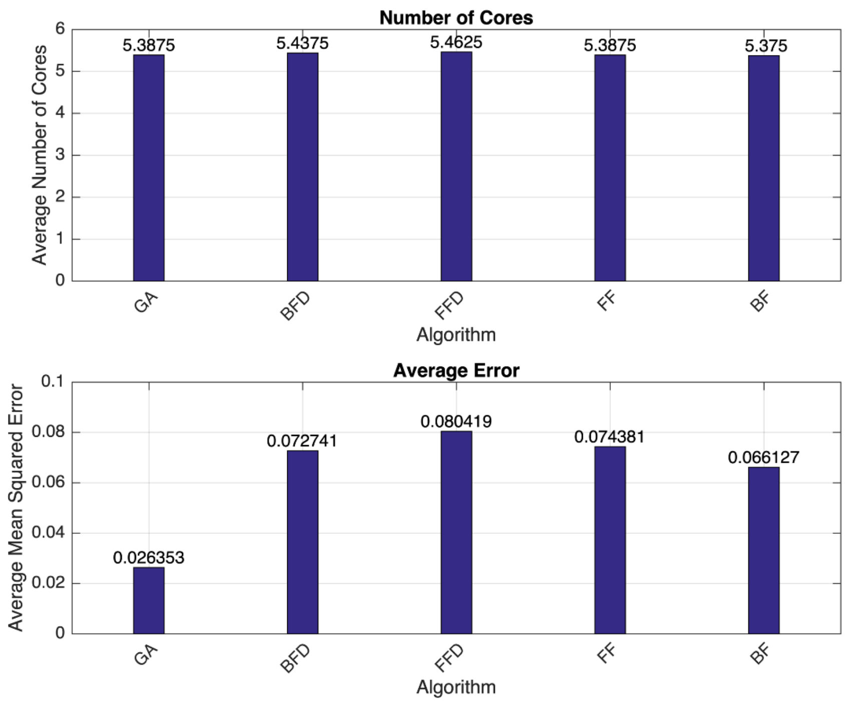

Figure 8. Results for the period range 10–500.

Finally, in Figure 9, the average number of cores used was six, with rounding, for all algorithms, where the average loss, again, varied by algorithm. Generally speaking, the ordered bin-packing algorithms tended to outperform their unsorted counterparts (BF and FF). For the period range 10–1000, the sorted bin-packing algorithms (BFD and FFD) had a smaller error rate when compared to their unsorted counterparts. The GA had 5.1% to 6.9% less error (a 65–71% difference) when compared to the bin-packing algorithms, which means that the GA outperformed the bin-packing heuristics in its ability to more closely match the expected scheduling uniformity across the same number of cores.

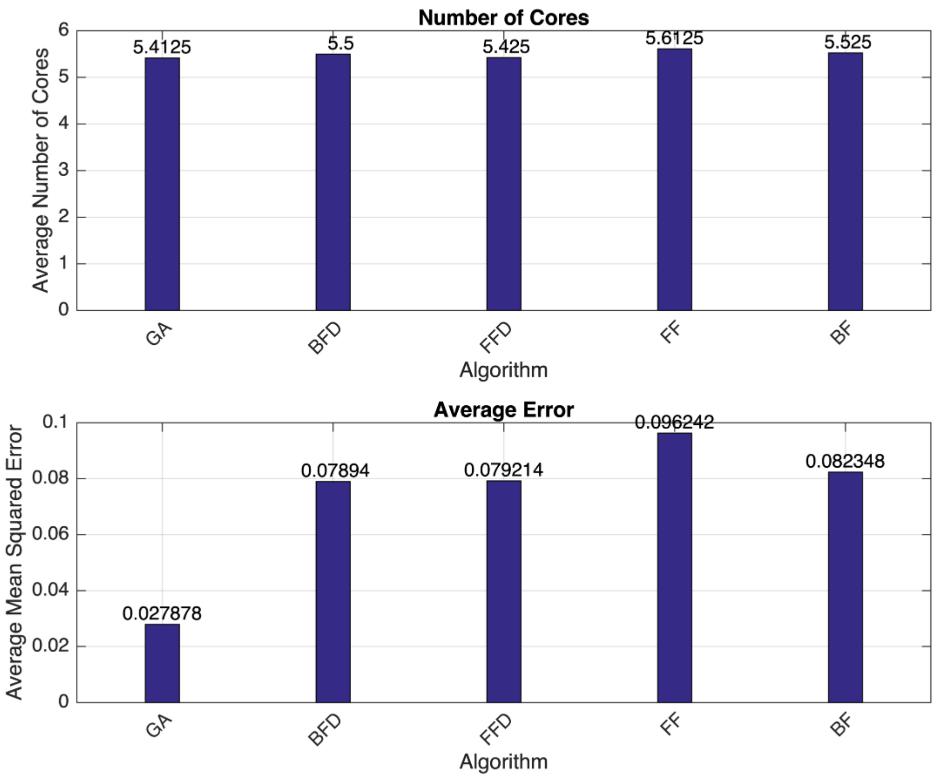

Figure 9. Results for the period range 10–1000.

6.3. Results Summary

In summary, Table 1 provides an assembly of the measures of performance (MSE and cores) for each period range tested. The average cores used was consistently six because of the manner of the data created. Starting with an 80% utilization and moving up to 100, each algorithm was able to schedule, on average, the tasks within the core count of 2, 4, 6, and 8 cores.

Table 1. Summary of results.

	10–100		**10–200**		**10–500**		**10–1000**	
	MSE	**Cores**	**MSE**	**Cores**	**MSE**	**Cores**	**MSE**	**Cores**
GA	1.9	6	2.7	6	2.6	6	2.8	6
BFD	6.3	6	6.2	6	7.3	6	7.9	6
FFD	6.2	6	6.5	6	8.0	6	7.9	6
FF	5.9	6	6.8	6	7.4	6	9.6	6
BF	5.6	6	6.5	6	6.6	6	8.2	6

The GA consistently provided a lower MSE with each period range as compared to the other scheduling heuristics. The bin-packing scheduling heuristics (BFD, FFD, BF, FF), however, increasingly performed worse with each period range. Each bin-packing heuristic, when compared to each other one, varied per period range tested. This data reveals that the GA is superior to the comparison algorithms because each bin-packing algorithm does not consider the entire task-to-partition-to-core schedule as a whole. The GA's objective function does consider the entire task-to-partition-to-core schedule through each generation of evolution, making it more uniform. The bin-packing heuristics can only test if the current task fits within the current allocation or not. The GA, on the other hand, can test the entire allocation scheme and evolves to the most optimal form.

7. Conclusions and Future Research

The aerospace and defense industries are facing an end-of-life production issue with legacy embedded uniprocessor systems. Current research does not address an automated way to move these legacy systems to a modern multicore architecture. These systems must also adhere to current safety and security requirements that create a complex problem which has been ignored. Safety and security requirements mandate a certain level of reserve or spare utilization spread uniformly across all cores. Bin-packing algorithms overload the front or back end of the cores, resulting in inefficient utilization. The aim of this research is to create a method to automate and optimize the migration of these uniprocessor systems to multicore systems. This method incorporates safety and security requirements which dictate uniformity across the cores. Important contributions of this research include the formation of a novel genetic algorithm, creation of a unique objective function, and a method to simulate and compare results. The simulation demonstrates the genetic algorithm is superior because it can be taught to create uniformity in utilization across the cores. The genetic algorithm has 1.8 to 2.5 times less error when compared to the bin-packing algorithms. Therefore, the genetic algorithm creates more uniform core utilization as compared to traditional bin-packing algorithms, which is essential when adding safety and security requirements. The result of this research provides a method for commercial, private, and defense industries to use a genetic algorithm designed to create a feasible two-level hierarchical schedule for real-time embedded multicore systems. The automated creation with inline analysis is demonstrated and will save money in terms of both cost and schedule for large legacy programs moving to multicore architectures.

This research created a hierarchical scheduling framework integrated with a genetic algorithm and demonstrated the efficacy of the approach. To provide benefit to the aerospace and defense industry, we must answer the question: how can this research be effectively applied to facilitate the migration of legacy embedded systems' real-time software tasks to multicore architectures? It offers a unified way to model, manage, and analyze the scheduling effects of virtualization in large complex systems of multicore architectures. Model-based systems engineering (MBSE) has garnered much attention in recent years. For legacy systems, MBSE integration is a gap. The genetic algorithm which creates the hierarchical scheduling arrangement can be modified to output into a universal modelling language. Then, it can be incorporated into a large system of systems model for integration into a real airborne platform. Future research endeavors will investigate and address MBSE integration.

Author Contributions: This paper was produced from B.W.'s dissertation. S.M. and A.E. are the coadvisors of B.W.'s committee. Both coadvisors worked with B.W. to edit and review the paper. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Raytheon Company.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Burns, A.; Davis, R.I. A Survey of Research into Mixed Criticality Systems. *ACM Comput. Surv.* **2018**, *50*, 1–37. [CrossRef]
2. Parkinson, P. Safety, Security and Multicore. *Adv. Syst. Saf.* **2010**, *10*, 215–232. [CrossRef]
3. Tămaş-Selicean, D.; Pop, P. Design Optimization of Mixed-Criticality Real-Time Embedded Systems. *Trans. Embed. Comput. Syst.* **2015**, *14*. [CrossRef]

4. Fu, N.; Shan, L.; Du, C.; Liu, Z.; Peng, H. Modeling and Verification of ARINC 653 Hierarchical Preemptive Scheduling. *Int. Arab J. Inf. Technol.* **2019**, *17*, 99–106. [CrossRef]
5. Gaska, T.; Werner, B.; Flagg, D. Applying Virtualization to Avionics Systems—The Integration Challenges. In Proceedings of the 2010 IEEE/AIAA 29th Digital Avionics Systems Conference (DASC), Salt Lake City, UT, USA, 3–7 October 2010; pp. 5.E.1-1–5.E.1-19.
6. Kotha, S.; Nasser, F. Virtualization-Enabled Industrial Grade Software Platforms [Industry Forum]. *IEEE Ind. Electron. Mag.* **2008**, *2*, 7–9. [CrossRef]
7. Main, C. Virtualization on Multicore for Industrial Real-Time Operating Systems [From Mind to Market]. *IEEE Ind. Electron. Mag.* **2010**, *4*, 4–6. [CrossRef]
8. Leroux, P.N.; Craig, R. Easing the Transition to Multi-Core Processors. *Inf. Q.* **2006**, *4*, 38–43.
9. Kleidermacher, D. *Hypervisors Thrive with Power Architecture*; Power Integrations, Inc.: San Jose, CA, USA, 2009; Available online: http://www.techdesign-forum.com/practice/technique/hypervisors-and-the-power-architecture/ (accessed on 3 September 2020).
10. Oshana, R. *DSP Software Development Techniques for Embedded and Real-Time Systems*; Newnes: Oxford, UK, 2006; p. 608.
11. Brandenburg, B.B. Scheduling and Locking in Multiprocessor Real-Time Operating Systems. Ph.D. Thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 2011.
12. Liu, C.L.; Layland, J.W. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM* **1973**, *20*. [CrossRef]
13. Dhall, S.K.; Liu, C.L. On a Real-Time Scheduling Problem. *Oper. Res.* **1978**, *26*, 127–140. [CrossRef]
14. Garey, M.R.; Johnson, D.S. *Computers and Intractability*; W H Freeman & Company: New York, NY, USA, 1979.
15. Monnier, Y.; Beauvais, J.P.; Deplanche, A.M. A genetic algorithm for scheduling tasks in a real-time distributed system. In Proceedings of the 24th EUROMICRO Conference (Cat. No. 98EX204), Vasteras, Sweden, 27 August 1998; pp. 708–714.
16. Lipari, G.; Bini, E. Resource partitioning among real-time applications. In Proceedings of the 15th Euromicro Conference on Real-Time Systems 2003, Porto, Portugal, 2–4 July 2013; pp. 151–158.
17. Regehr, J.; Reid, A.; Webb, K.; Parker, M.; Lepreau, J. Evolving real-time systems using hierarchical scheduling and concurrency analysis. In Proceedings of the 24th Real-Time System Symposium (RTSS 2003), Cancun, Mexico, 5 December 2003; pp. 25–36.
18. Regehr, J.D. Using Hierarchical Scheduling to Support Soft Real-Time Applications in General-Purpose Operating Systems. Ph.D. Thesis, University of Virginia, Charlottesville, VA, USA, May 2001.
19. Lipari, G.; Bini, E. A methodology for designing hierarchical scheduling systems. *J. Embed. Comput.* **2005**, *1*, 257–269.
20. Asberg, M. On Hierarchical Scheduling in VxWorks. Master's Thesis, Department of Computer Science and Electronics, Malardalen University, Vasteras, Sweden, June 2008.
21. Behnam, M.; Nolte, T.; Shin, I.; Åsberg, M.; Bril, R. Towards Hierarchical Scheduling in VxWorks. In Proceeding of the Fourth International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, Prague, Czech Republic, 1 July 2008; pp. 63–72.
22. Nolte, T.; Behnam, M.; Asberg, M.; Bril, R.J. Hierarchical Scheduling of Complex Embedded Real-Time Systems. Available online: https://www.researchgate.net/publication/241877785_Hierarchical_scheduling_of_complex_embedded_real-time_systems (accessed on 3 September 2020).
23. Asberg, M.; Nolte, T.; Kato, S. Towards Hierarchical Scheduling In Linux/Multi-Core Platform. In Proceedings of the Factory Automation (ETFA 2010), Bilbao, Spain, 13–16 September 2010; pp. 1–4.
24. Cullmann, C.; Ferdinand, C.; Gebhard, G.; Grund, D.; Maiza, C.; Reineke, J.; Triquet, B.; Wilhelm, R. Predictability considerations in the design of multi-core embedded systems. In Proceedings of the Embedded Real Time Software and Systems 2010, Touluse, France, 17–21 May 2010; pp. 36–42.
25. Hilbrich, R.; Goltz, H.-J. Model-based Generation of Static Schedules for Safety Critical Multi-Core Systems in the Avionics Domain. In Proceedings of the ICSE'11: International Conference on Software Engineering Waikiki, Honolulu HI, USA, 21 May 2011; Association for Computing Machinery: New York, NY, USA, 2011; pp. 9–16.

26. Xu, C.; Gamage, S.; Rao, P.N.; Kangarlou, A.; Kompella, R.R.; Xu, D. Vslicer: Latency-Aware Virtual Machine Scheduling Via Differentiated-Frequency Cpu Slicing. In *HDC'12: Proceedings of the21International Symposium on High Performance Parallel and Distributed Computing, Delft, The Netherlands, June 2012*; Association for Computing Macinery: New York, NY, USA, 2012; pp. 3–14. [CrossRef]
27. Nemati, F.; Kraft, J.; Nolte, T. Towards Migrating Legacy Real-Time Systems to Multi-Core Platforms. In Proceedings of the Emerging Technologies and Factory Automation (IEEE 2008), Hamburg, Germany, 15–18 September 2008; pp. 717–720.
28. Nemati, F.; Kraft, J.; Nolte, T. A Framework for Real-Time Systems Migration to Multi-Cores. Available online: https://www.researchgate.net/publication/241877785_Hierarchical_scheduling_of_complex_embedded_real-time_systems (accessed on 3 September 2020).
29. Nemati, F.; Behnam, M.; Nolte, T. Efficiently migrating real-time systems to multi-cores. In Proceedings of the Conference on Emerging Technologies & Factory Automation (2009 IEEE), Mallorca, Spain, 22–25 September 2009.
30. Kwon, Y.; Kim, C.; Maeng, S.; Huh, J. Virtualizing Performance Asymmetric Multi-Core Systems. In Proceedings of the 38th Annual International Symposium Computer Architecture (ISCA 2011), San Jose, CA, USA, 4–8 June 2011; pp. 45–56.
31. Masrur, A.; Pfeuffer, T.; Geier, M.; Drossler, S.; Chakraborty, S. Designing VM schedulers for embedded real-time applications. In Proceedings of the 2011 Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Taipei, Taiwan, 9–14 October 2011; pp. 29–38. [CrossRef]
32. Brownlee, J. *Clever Algorithms*; Lulu Press Inc.: Morissville, NC, USA, 2011; p. 438.
33. Holland, J.H. *Adaptation in Natural and Artificial Systems*; MIT Press: Cambridge, MA, USA, 1975.
34. Davis, L. *Handbook of Genetic Algorithms*; Van Nostrand Reinhold: New York, NY, USA, 1991.
35. Melanie, M. *An Introduction to Genetic Algorithms*; MIT Press: Cambridge, MA, USA, 1999.
36. Jensen, M.T. Generating robust and flexible job shop schedules using genetic algorithms. *IEEE Trans. Evol. Comput.* **2003**, *7*, 275–288. [CrossRef]
37. Chryssolouris, G.; Subramaniam, V. Dynamic scheduling of manufacturing job shops using genetic algorithms. *J. Intell. Manuf.* **2001**, *12*, 281–293. [CrossRef]
38. Wall, M.B. A Genetic Algorithm for Resource-Constrained Scheduling. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1996.
39. Pezzella, F.; Morganti, G.; Ciaschetti, G. A genetic algorithm for the Flexible Job-shop Scheduling Problem. *Comput. Oper. Res.* **2008**, *35*, 3202–3212. [CrossRef]
40. Mo, Z.; Wu, G.; He, Y.; Liu, H. Quantum Genetic Algorithm for Scheduling Jobs on Computational Grids. In Proceedings of the 2010 International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), Changsha City, China, 13–14 March 2010; pp. 964–967.
41. Nguyen, S.; Zhang, M.; Johnston, M.; Tan, K.C. Automatic Design of Scheduling Policies for Dynamic Multi-objective Job Shop Scheduling via Cooperative Coevolution Genetic Programming. *IEEE Trans. Evol. Comput.* **2014**, *18*, 193–208. [CrossRef]
42. Montana, D.; Brinn, M.; Moore, S.; Bidwell, G. Genetic algorithms for complex, real-time scheduling. In Proceedings of the (SMC 98 Conference Proceedings) 1998 IEEE International Conference on Systems, Man and Cybernetics, San Diego, CA, USA, 14 October 1998; pp. 2213–2218.
43. Mahmood, A. A Hybrid Genetic Algorithm for Task Scheduling in Multiprocessor Real-Time Systems. *Stud. Inform. Control* **2000**, *9*, 207–218.
44. Oh, J.; Wu, C. Genetic-algorithm-based real-time task scheduling with multiple goals. *J. Syst. Softw.* **2004**, *71*. [CrossRef]
45. Miryani, M.R.; Naghibzadeh, M. Hard Real-Time Multiobjective Scheduling in Heterogeneous Systems Using Genetic Algorithms. *Comput. Conf.* **2009**, 437–445. [CrossRef]
46. Page, A.J.; Naughton, T.J. Dynamic Task Scheduling using Genetic Algorithms for Heterogeneous Distributed Computing. In Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, Denver, CO, USA, 4–8 April 2005; pp. 189–189a.
47. Carretero, J.; Xhafa, F. Genetic algorithm based schedulers for grid computing systems. *Int. J. Innov. Comput.* **2007**, *3*, 5.

48. Yoo, M.; Gen, M. Scheduling algorithm for real-time tasks using multiobjective hybrid genetic algorithm in heterogeneous multiprocessors system. *Comput. Oper. Res.* **2007**, *34*, 3084–3098. [CrossRef]
49. Rahmani, A.M.; Vahedi, M.A. A novel Task Scheduling in Multiprocessor Systems with Genetic Algorithm by using elitism stepping method. In Proceedings of the International Conference of Service Operations and Logistics, and Informations (IEEE 2008), Beijing, China, 12–15 October 2008.
50. Kang, Y.; Zhang, Z.; Chen, P. An Activity-Based Genetic Algorithm Approach to Multiprocessor Scheduling. In Proceedings of the 2011 Seventh International Conference on Natural Computation (ICNC), Shanghai, China, 26–28 July 2011; pp. 1048–1052.
51. Kaur, R.; Singh, G. Genetic Algorithm Solution for Scheduling Jobs In Multiprocessor Environment. In Proceedings of the 2012 Annual IEEE India Conference (INDICON), Kochi, India, 7–9 December 2012; pp. 968–973.
52. Montana, D.J.; Bidwell, G.; Moore, S. Using genetic algorithms for complex, real-time scheduling applications. *NOMS* **1998**, *1*, 245–248. [CrossRef]
53. Yepez, J.; Guardia, J.; Velasco, M.; Ayza, J.; Castane, R.; Marti, P.; Fuertes, J.M. CICLIC: A Tool to Generate Feasible Cyclic Schedules. In Proceedings of the 2006 IEEE International Workshop on Factory Communication Systems, Torino, Italy, 28–30 June 2006; pp. 399–404.
54. Gracioli, G.; Fröhlich, A.A. CAP: Color-aware task partitioning for multicore real-time applications. *ETFA* **2014**, 1–8. [CrossRef]
55. Davis, R.I.; Burns, A. Priority Assignment for Global Fixed Priority Pre-Emptive Scheduling in Multiprocessor Real-Time Systems. *Symposium* **2009**, 398–409. [CrossRef]
56. Shehzad, M.N.; Deplanche, A.M.; Trinquet, Y. Efficient data generation for the testing of real-time multiprocessor scheduling algorithms. *Prz. Elektrotechniczny* **2014**, *90*. [CrossRef]
57. Craveiro, J.P.G.C. Real-Time Scheduling in Multicore: Time-and Space-Partitioned Architectures. Ph.D. Thesis, Universidade De Lisboa, Lisbon, Portugal, 2014.
58. Schreiber, E.L.; Korf, R.E. Improved Bin Completion for Optimal Bin Packing and Number Partitioning. In Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, Beijing, China, 3–9 August 2013; pp. 651–658.
59. Belwal, C.; Cheng, A.M.K. Generating Bounded Task Periods for Experimental Schedulability Analysis. In Proceedings of the 2011 IFIP 9th International Conference on Embedded and Ubiquitous Computing (EUC), Melbourne, VIC, Australia, 24–26 October 2011; pp. 249–254.
60. Niz, D.d.; Rajkumar, R. Partitioning bin-packing algorithms for distributed real-time systems. *IJES* **2006**, *2*, 196–208. [CrossRef]