US 20260099372A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2026/0099372 A1**

MATL et al. (43) **Pub. Date: Apr. 9, 2026**

(54) **METHOD TO EXECUTE TIME-TRIGGERED TASKS IN A COMPUTER SYSTEM WITH FLEXIBLE EXECUTION INTERVALS**

(71) Applicant: **TTTech Computertechnik AG**, Vienna (AT)

(72) Inventors: **Piotr MATL**, Vienna (AT); **Silviu CRACIUNAS**, Vienna (AT); **Ramon SERNA OLIVER**, Vienna (AT)

(21) Appl. No.: **19/324,665**

(22) Filed: **Sep. 10, 2025**

(30) **Foreign Application Priority Data**

Oct. 8, 2024 (EP) 24205298.3

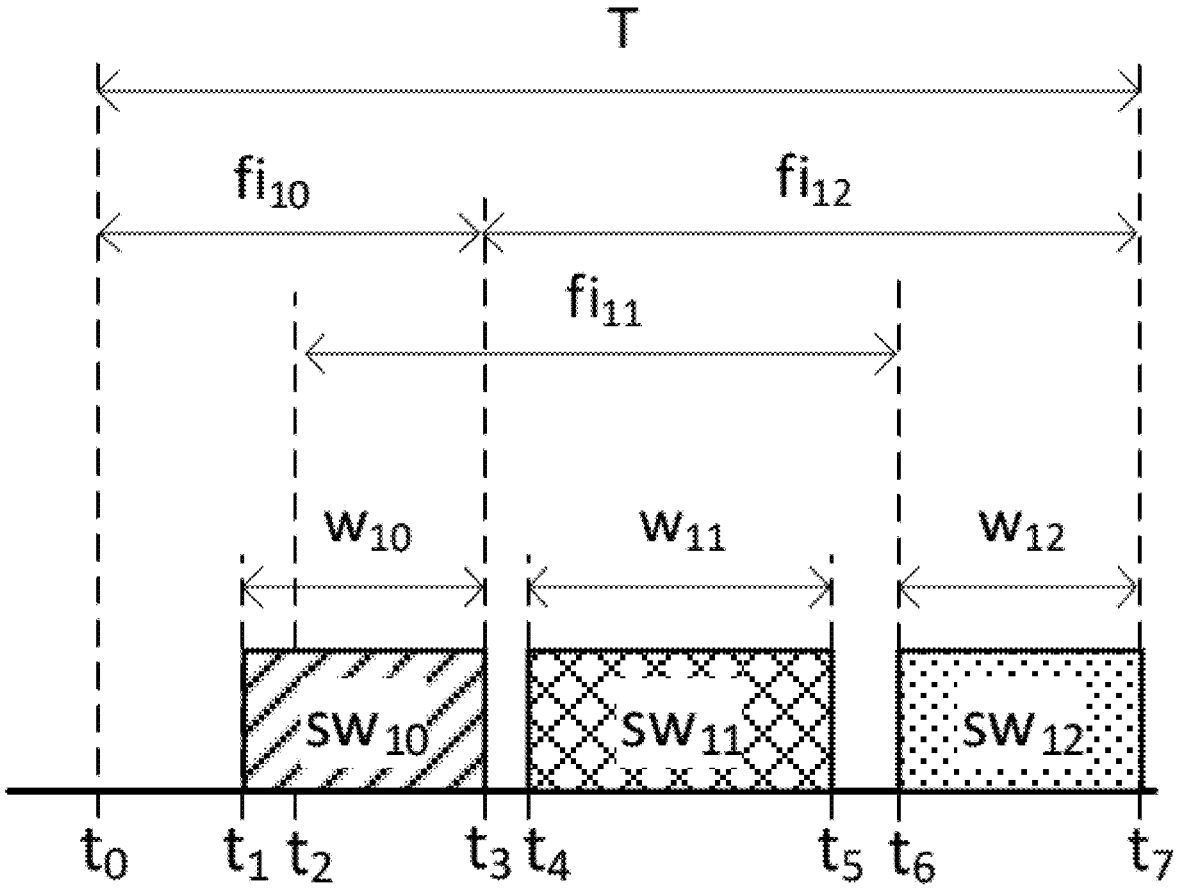
Publication Classification

(51) **Int. Cl.**
G06F 9/50 (2006.01)

(52) **U.S. Cl.**
CPC ***G06F 9/5027*** (2013.01)

(57) **ABSTRACT**

A method to execute jobs of a set of tasks in a computer system according to a global time-triggered schedule, wherein the computer system comprises one or more cores on one or more central processing units of the computer system. The global time-triggered schedule includes one time-triggered schedule, "core schedule", for each core, wherein the tasks of the set of tasks to be executed on a specific core are executed on the core according to the core schedule of the core. The core schedule specifies for each job a flexibility interval, wherein the flexibility of a job defines a time interval during which the job can be executed. Runtime dispatching means select and launch jobs for execution according to the global time-triggered schedule, taking into account for each core all flexibility intervals as specified in the core schedule.



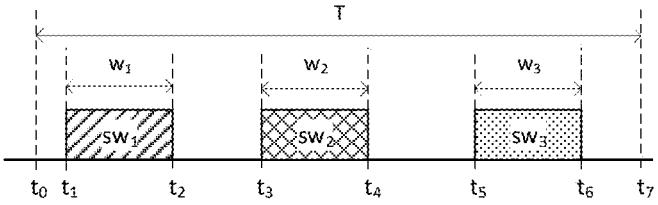


Figure 1

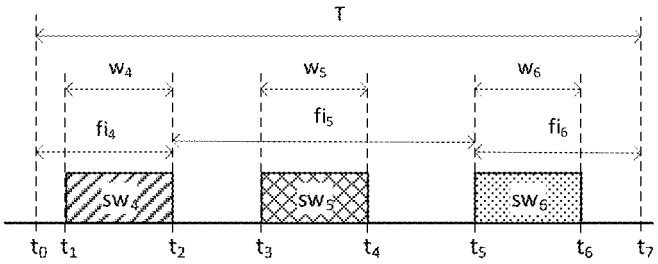


Figure 2

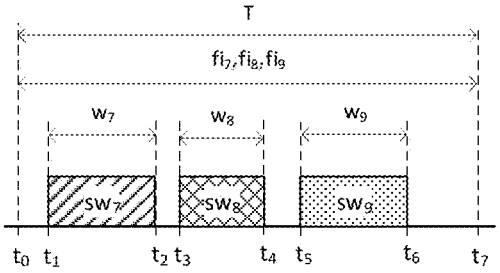


Figure 3

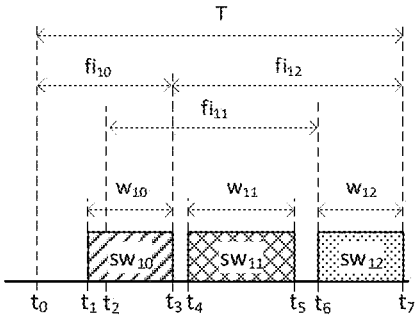


Figure 4

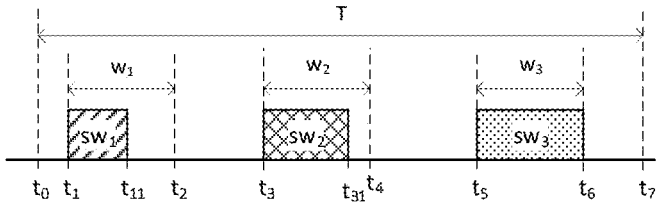


Figure 5

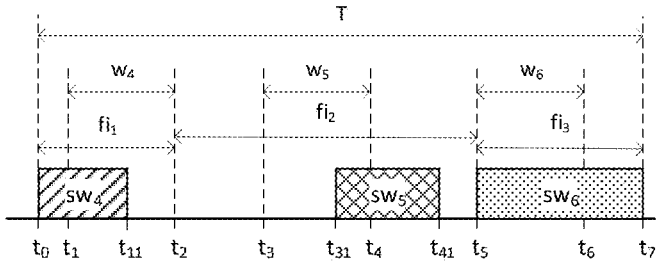


Figure 6

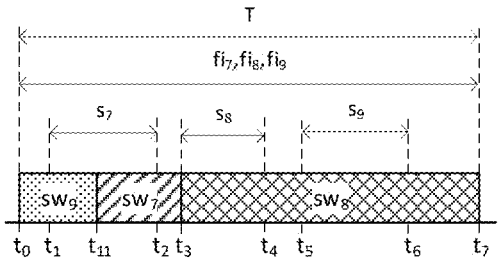


Figure 7

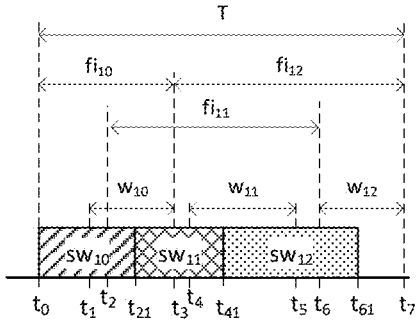


Figure 8

METHOD TO EXECUTE TIME-TRIGGERED TASKS IN A COMPUTER SYSTEM WITH FLEXIBLE EXECUTION INTERVALS

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority to European Patent Application No. 24205298.3, filed Oct. 8, 2024, which is incorporated herein by reference in its entirety.

TECHNICAL FIELD

[0002] The present invention relates to a method to execute jobs of a set of tasks in a computer system according to a global time-triggered schedule,

[0003] wherein the computer system comprises one or more cores on one or more central processing units of the computer system,

[0004] wherein said global time-triggered schedule comprises one time-triggered schedule, "core schedule", for each of said cores, wherein the tasks of the set of tasks to be executed on a specific core are executed on said core according to the core schedule of said core,

[0005] wherein the global time-triggered schedule specifies a cycle with a cycle length for the set of tasks,

[0006] wherein the core schedule of a core specifies for each task of the set of tasks, which is to be executed on said core:

[0007] an execution interval for each task instance, "job", of the task, during which execution interval the job can be executed, wherein each execution interval comprises a release point, at which the execution interval starts, and a completion point, at which the execution interval ends, wherein the duration of an execution interval is the duration of the time span between the release point and the completion point of the execution interval, wherein the execution interval of a job defines a time interval during which the job has a guarantee to be executed on the core, and

[0008] a period according to which period the jobs of said task are executed on the core,

[0009] wherein the execution interval of each job of the task is iterated cyclically according to the cycle, and

[0010] wherein the execution intervals of any two jobs of the same and of different tasks do not overlap, and

[0011] wherein the cycle is a common multiple of all periods of all tasks in the set of tasks.

TECHNICAL BACKGROUND

[0012] The execution of periodic jobs of software tasks following a time-triggered paradigm has been studied in literature and employed in industry for many years. Among its benefits, stands out the deterministic timely execution of the jobs of software tasks in a cyclic manner, generally driven by an offline-computed schedule (the tt-schedule) allowing a correctness-by-design methodology. Moreover, tt-schedules can be computed based on optimization objectives that are hard to reach with other paradigms, like minimizing job-level jitter or fulfilling complex job interdependencies.

[0013] However, it is also known that the time-triggered paradigm offers little to no flexibility in terms of dynamically adapting to changes in the runtime behavior of software jobs. This lack of adaptation may be an asset for critical systems requiring an exact and deterministic behavior over time, but it may also reduce the capability to maximize the usage of resources in systems with less stringent determinism requirements.

[0014] Prior to computing a tt-schedule, software tasks must be characterized, particularly with respect to their timeliness requirements, like required execution time and execution rate (i.e., period). A common objective of the time-triggered paradigm is to compute the tt-schedule based on the worst-case timeliness bounds, like the worst-case execution time and lowest execution rate. Once the tt-schedule is constructed, it is cyclically enforced at runtime by a dispatching mechanism. Accounting for the worst-case guarantees that any change in the behavior of the jobs of a task during runtime will not exceed the system capacity, and hence the execution correctness remains guaranteed.

[0015] However, in modern computer systems, particularly automotive computer systems, it is typically an objective to maximize the utilization of hardware components, like CPU and memory since any unused resource would be preferably scaled down at design time to reduce production costs and overall power consumption.

[0016] These two objectives often conflict with each other and cannot be trivially solved. Indeed, adding flexibility to the time-triggered paradigm enabling the adaptation to runtime changes in the execution timeliness of the periodic jobs of tasks while conserving determinism and correctness guarantees is a challenge.

Time-Triggered Scheduling

[0017] Existing time-triggered scheduling methods plan the execution of software tasks in defined CPU cores by computing one or multiple execution intervals for each periodic job of said software task in one or multiple of said CPU core(s), in accordance with a defined set of scheduling constraints.

[0018] An execution interval comprises a start time and a duration for which the assigned CPU core(s) will exclusively execute the assigned job of said software tasks.

[0019] A computed tt-schedule is typically represented in the form of a timetable containing a list of computed intervals with the assignment of jobs and CPU core(s).

[0020] The computation of a tt-schedule is based on defined scheduling constraints determining the correct relation between execution intervals conforming a valid tt-schedule. For example, an execution interval cannot be assigned to multiple jobs, as it would result on a single resource (the CPU core) required to execute the instructions of two jobs simultaneously, which is not feasible. As another example, different execution intervals on the same CPU core cannot overlap, as this would again result in multiple jobs planned to execute simultaneously on the same CPU core, hence invalidating the guaranteed reservation of execution time for each individual job.

TT Scheduling Constraints

[0021] A computer system comprises one or more hosts, each of which comprises one or more CPU cores.

[0022] A software application consists of one or more periodic software components, or tasks. Each task is characterized by their period, an earliest release time, and a

deadline-A task is cyclically executed on a core according to its period, bearing a sequence of task execution instances, or jobs.

[0023] A tt-schedule is computed according to general correctness constraints for TT scheduled found in literature, e.g., [1], and summarized here for completeness. The computation of a tt-schedule relates to computing execution intervals for each periodic job of a software task within a defined schedule cycle, wherein said computation conforms to defined scheduling constraints, including, but not limited to:

[0024] Start constraint: The start of the execution of any job of a software task sw_i must be after the given earliest release time e; in each period p_i.

[0025] Deadline constraint: The end of the execution of any job of a software task sw_i must be before the given deadline d_i in each period p_i.

[0026] Non-overlap constraint: No two jobs of any tasks can execute at the same time on the same core.

[0027] Dependency constraints: Other requirements further restricting the interrelation of software tasks introduce additional dependency constraints when computing the tt-schedule. For example, in some cases functional correctness requires that the execution of a job of a first software task shall occur, in all cycles, following the execution of a job of a second software task. In other cases, the order of execution of jobs of a group of software tasks may conform to a directed acyclic graph (DAG), whereby within a certain maximum interval of time the jobs of the first and last software tasks of the DAG shall complete their execution. These additional dependency constraints may be categorized as simple dependency constraints, or complex dependency constraints, depending on the complexity of said interrelation.

[0028] Tasks may have either simple or complex dependencies. A dependency comprises two or more tasks, wherein a defined order of execution between jobs of said tasks are specified, wherein

[0029] in a simple dependency all of said comprised tasks have the same period, whereas

[0030] in a complex dependency at least two of said comprised tasks have different period.

[0031] In addition, simple and complex dependencies relate to a maximum latency constraint, wherein said maximum latency can be one of the following

[0032] a maximum response time, specifying that for all jobs of the first task in the set of defined tasks, there exists a trace of successor jobs following a defined order, through all other intermediary jobs of tasks in the set, such that the trace does not take longer than said response time,

[0033] a maximum data age, specifying that for all jobs of the last task in the set of tasks according to the defined order, there exists a trace from some jobs of the first task in the set of tasks, going through jobs of all other intermediary tasks in the set, whereby said trace does not take longer than said data age.

[0034] Jitter constraint: For every job of a task, there is a non-negative delay between the scheduled start and the earliest release time; the jitter constraint defines a maximum permitted difference between said delays for any pair of jobs of a task.

[0035] It is known that the construction of a tt-schedule is a computationally intense operation that cannot be solved optimally within reasonable time for large systems [2, 3]. Approximative heuristic algorithms have been used in the past and perform reasonably well on average cases [4]

Runtime Execution Variability

[0036] In some cases, the sequence of jobs of a task may exhibit different run-time execution requirements along their cyclic execution iterations, due for example to differences over time in the data they need to process. This run-time variability can be addressed when computing a tt-schedule by assuming the worst-case runtime (worst-case computation time, or WCET) for each job of a software task, hence planning the amount and length of execution intervals sufficient to satisfy said worst-case behavior (e.g., largest execution path).

[0037] An advantage of a time-triggered scheduling paradigm is the deterministic and timely behavior of the executed jobs of software tasks during runtime, regardless of their cyclic variation in the execution requirements, typically resulting in only minimal deviations between cycles (e.g., due to unavoidable runtime inaccuracy introduced by uncertainties of the underlying hardware and software components).

Runtime Adaptability

[0038] In the general case, the construction during design-time of the tt-schedule prevents exercising runtime adaptations to the dynamic variability of the jobs of software tasks and can be a disadvantage of the time-triggered scheduling paradigm. For example, when a job of a software task does not require a complete execution interval within a cycle, the remaining time (slack) cannot be easily reassigned to the jobs of the next software tasks in the tt-schedule.

[0039] Another related disadvantage of a time-triggered scheduling paradigm is the overprovisioning caused by assuming a worst-case bound for the timeliness of software tasks, in particular a WCET, when computing the timetable, hence assigning larger execution intervals than on average required during execution, since during runtime, jobs often require less execution time than the WCET of their respective task.

[0040] Some existing methods address the negative effects of overprovisioning due to the WCET by computing a tt-schedule with consideration of lower bounds, for example the average execution time, instead of the WCET, for certain tasks [5], whereby if jobs of said tasks exceed their assigned execution interval in any of its cycles, they may be interrupted and not accomplish their execution in the current cycle, or in some cases, jeopardize jobs of other software tasks by exceeding their planned execution interval.

[0041] Other existing methods allow the computer system to exchange its configuration during runtime among a set of alternative precomputed tt-schedules (e.g., referred as "modes") [6]. These methods try to estimate at design time the variability that will be observed during runtime and compute dedicated tt-schedules adapted to these conditions. This adaptation of the configuration is commonly known as performing a system mode change. In these methods, the number of supported modes are limited by two factors: on one hand the number of possible system states (observable

variability) resulting on the need of a different tt-schedule grows exponentially with the size of the system (e.g., number of CPU cores, number of software tasks, number of constraints), and hence only a fraction of them can be feasibly stored in the computer system for future use (e.g., due to limited memory capacity); on the other hand, a system mode change cannot be applied instantaneously and may require complex algorithms to guarantee system stability at runtime during the transition, hence adding latency and limiting the adaptability chanced in a deployed running system.

[0042] Yet other existing methods include mechanisms to dynamically modify the computed tt-schedule during runtime [9, 10], in response to an actual state of the system. These methods are limited by the computation complexity of the constraints involved in the fast (re) computation of a new tt-schedule, typically within one, or few, execution cycles. Hence, the employed algorithms cannot handle large numbers of constraints, or fulfill all constraints within a sufficiently fast response time. In addition, safety and certification concerns arise when dynamic changes may negatively affect the timely execution of jobs related to safety-critical tasks, for example those in deployed automotive or industrial applications. Therefore, the modification of timetables at runtime has a limited use in real-world use-cases.

Operating System/Runtime System Dispatcher

[0043] An active module, typically a software component, is provided in the computer system, which module orchestrates the execution of software tasks by selecting, at defined instants of time, jobs of tasks from a set of ready jobs and dispatching them for execution in one or more of the CPU cores in the computer system. This active module, the dispatcher (or dispatching means), is the runtime component enforcing the decisions made by the scheduler and may be implemented in a computer system in different forms, for example as a stand-alone runtime software component, as a module of an operating system, or as a high-level software stack.

[0044] Usually, one such active module is provided. However, it may also be possible that two or more such active modules are provided.

[0045] In a computer system implementing a time-triggered paradigm, the dispatcher selects jobs for execution based on the tt-schedule. Once a job is dispatched, the dispatcher waits until the end of the execution interval of said job and proceeds to select the next job following the order and timing defined in the tt-schedule (i.e., timetable).

[0046] In computer systems implementing other scheduling paradigms, the dispatcher may select jobs of software tasks for execution based on defined criteria. For example, in computer systems implementing an earlier deadline first (EDF) paradigm the dispatcher selects for execution the jobs with the nearest deadline from the set of ready jobs. In contrast, in a computer system implementing a fixed priority (FP) paradigm, software tasks are characterized by a defined and fixed priority, whereby the dispatcher selects for execution the jobs of software task with the highest defined priority from the set of ready jobs. When the dispatched job yields, or new jobs become ready, the dispatcher may select and launch a new job based on the same criteria.

SUMMARY OF THE INVENTION

[0047] It is an object of the invention to optimize the usage of resources, in particular of CPU time and/or core time in a computer system in which jobs of periodic tasks are executed according to a time-triggered schedule.

[0048] This is achieved with a method according to claim **1**, wherein the core schedule specifies for each job a flexibility interval, wherein the duration of the flexibility interval of a job is equal or longer than the execution interval of said job, wherein

[0049] at least the flexibility interval of one job is longer than the execution interval of said job, wherein

[0050] the flexibility interval of a job starts no later than the release point (start point, start/release instant) of the execution interval of said job, and wherein

[0051] the flexibility interval of a job ends no sooner than the completion point (end point, end/completion instant) of the execution interval of said job, wherein

[0052] the flexibility of a job defines a time interval during which said job can be executed, and wherein

[0053] runtime dispatching means are provided, which select and launch jobs for execution according to the global time-triggered schedule, taking into account for each core all flexibility intervals as specified in the core schedule.

[0054] The periodic execution of tasks within the schedule cycle comprises one or more executions of task instances, or jobs, of said tasks represented by one said element in the timetable, wherein each job corresponds to an execution of the task according to its periodicity, repeated cyclically according to the schedule cycle. If the task period is equal to the schedule cycle, there is one single job of said task within the schedule cycle, whereas if the schedule cycle is larger than the period, there are multiple jobs of said task according to the relation between the period and the schedule cycle. Note that the length of the schedule cycle preferably is a common multiple of the period of all tasks, for example, the least common multiple. If, for example, the period of a task is 10 and the cycle is 20, there will be two instances of the task ("jobs") within the cycle. Each of them will have its own intervals within their respective period interval (from 0 . . . 10, and 10 . . . 20), and the flexibility intervals may even be of different length and belong to a different one of the three subsets, which subsets will be explained below.

[0055] According to the invention, flexibility intervals are provided for each job, whereby the flexibility interval of a job fully overlaps the execution interval of this job and is equal or larger than its execution interval. This makes it possible for the runtime dispatching means to organize the execution of jobs more flexibly in terms of time and thus to make better use of the available resources, in particular to allow a better usage of the time of each core.

[0056] As mentioned, the invention requires that at least the flexibility interval of one job is longer than the execution interval of said job. For an optimal effect of the invention, it is advantageous that the flexibility intervals of several jobs, in particular of a plurality of jobs, preferably of all jobs are longer than the corresponding execution intervals.

[0057] In a general case, a method to compute tt schedules relates to deciding a valid placement on a timeline for each execution interval of each job to be scheduled, subject to defined scheduling constraints defining the schedulability bounds, bounds, for the correct placement of said intervals. Once a tt-schedule is computed, said constraints are implicitly fulfilled by the provided execution intervals conforming the schedule. The placement of the execution interval of

each job is bounded by the limits imposed by the defined scheduling constraints. However, in the computation process, multiple decisions are possible within said bounds, resulting in alternative but equivalently correct tt-schedules.

[0058] For example, a periodic job subject to a deadline constraint may be scheduled arbitrarily within the interval bounded by the period start and the relative deadline, provided periodicity and deadline are the only constraints to be fulfilled. By exercising a decision to fix the execution interval of the job, the scheduling process selects one of multiple possible correct execution intervals for said job.

[0059] In some cases, early decisions made during the computation of a tt-schedule affect later decisions related to other jobs. For example, when jobs are subject to dependency constraints, any decision affecting the execution interval of a precedent job affects the decision of a successor job, since the dependency mandates that the successor job can only be scheduled after the execution interval of the predecessor job, hence tightening the lower bound to the execution interval of the successor job.

[0060] Some methods to compute tt-schedules apply a simple policy to select the start of execution intervals base on the lower bound when deciding the placement of the execution interval of each job, i.e., scheduling jobs as soon as possible. Alternatively, other methods apply a policy to select the end of the execution interval based on the upper bound when deciding the placement of the execution interval of each job, i.e., scheduling jobs as late as possible. More complex methods select arbitrary values between said bounds when deciding the placement of the execution interval of each job, based on additional defined criteria.

[0061] It may be provided that a method to compute tt-schedules is extended to keep track and update the schedulability bounds for each job, by iteratively adjusting the upper and lower bounds of all jobs upon each scheduling decision. It may be also provided that said extended method, after computing the tt-schedule, additionally computes flexibility intervals for each job based on said upper and lower schedulability bounds, wherein for each job, the start of its flexibility interval corresponds to the lower schedulability bound, and the end of its flexibility interval corresponds to the upper schedulability bound.

[0062] It may also be provided a method to derive schedulability bounds based on an existing tt-schedule and the set of scheduling constraints, wherein said method computes upper and lower bounds for each execution interval, subject to the set of defined scheduling constraints. For example, an iterative method may initiate the flexibility interval of a first job from the tt-schedule to the interval corresponding to its execution interval and iteratively increase it while checking that the set of defined constraints are still fulfilled. Once at least on constraint is not fulfilled, the flexibility interval for said first job is fixed to that of the previous iteration and a next job is selected as first job until all jobs have been provided a flexibility interval.

[0063] It may be provided that among the jobs of those tasks of the set of tasks that are to be executed on the core at least one first subset of jobs is provided, wherein for the jobs of said at least one first subset the core schedule specifies flexibility intervals which do not overlap with each other, wherein at least one of the flexibility intervals of a job is longer than the execution of said job.

[0064] This first case, in which jobs which comprise a flexibility interval are jobs with flexibility intervals following the rules of said first subset, maximally simplifies the runtime behavior of the dispatcher.

[0065] Alternative or in addition it may be provided that among those tasks of the set of tasks that are to be executed on the core at least one second subset of jobs is provided, wherein for the jobs of said at least one second subset the core schedule specifies flexibility intervals so that the flexibility intervals of two or more jobs of said at least one second subset of jobs are identical, i.e., for said two or more jobs of said at least one second subset the start instant of their flexibility intervals is the same instant and the end instant of their flexibility intervals is the same instant. For example, at least one of the flexibility intervals of a job is longer than the execution of said job.

[0066] This second case, in which jobs which comprise a flexibility interval are jobs with flexibility intervals following the rules of said second subset, also allows a simplified runtime behavior of the dispatcher, as it may be provided that the jobs of this second subset can be executed in any arbitrary order within said interval.

[0067] Alternative or in addition it may be provided that among those tasks of the set of tasks that are to be executed on the core at least one third subset of jobs is provided, wherein the flexibility intervals of any two jobs of said at least one third subset partially overlap. For example, at least one of the flexibility intervals of a job is longer than the execution of said job

[0068] This third case, in which jobs which comprise a flexibility interval are jobs with flexibility intervals following the rules of said third subset, adds the most complexity to the runtime dispatching means.

[0069] In this context it is noted that the subsets as described are self-contained, meaning that for those jobs with overlapping intervals the overlap must be between jobs of the same set (i.e., not with a job which is in one of the other sets).

[0070] The non-overlapping subset (first subset, subset 1) is the most restrictive subset: for a job to be in this subset its flexibility intervals shall not overlap with any flexibility interval of any other job of this subset. If there is an overlap between flexibility intervals of two jobs, both overlapping jobs are not part of the first subset.

[0071] Next is the second subset (subset 2) with identical flexibility intervals. Flexibility intervals of a jobs of this subset shall overlap only with flexibility intervals of other jobs with identical flexibility interval. If there is an overlap with a non-identical flexibility interval, then again both jobs cannot be in the second subset.

[0072] Any other combination of flexibility intervals refers to tasks of the third subset.

[0073] In the invention, tasks from at least one subset must be present, wherein said at least one present subset contains at least one flexibility interval which is longer than the corresponding execution interval. However, jobs from two or even three subsets may also be present. The dispatching means must be configured to be able to handle jobs from the subsets present. Typically, a dispatcher is provided which can handle jobs from all subsets, but it is also possible for the dispatching means to comprise a separate dispatcher for each of the present subsets.

[0074] It should be noted that there can be more subsets of each type. For example, there could be two different subsets of the "type" subsets 1, one with 3 jobs, one with 5, wherein all 3 jobs of the first subset 1 overlap with each other, and

all 5 jobs of the second subset 1 overlap with each other, but none of the 3 jobs overlap with none of the 5 jobs.

[0075] Regarding a dispatching policy applied by the dispatching means, the different possible cases (case 1-case 3) connected to the subsets of jobs (subset 1-subset 3) described above will be described in the following.

[0076] For completeness, a "case 0" is described here, which may occur, since jobs can still be present which have "rigid" execution intervals and no flexibility intervals, which must also be processed by the dispatching means. The dispatching policy in this case is as follows: the runtime dispatching means will iterate over the timetable while keeping track of the progress of time in relation to the cycle. The dispatching means (also called "dispatcher") will compare the current time with the instants of time defining the start and end of each execution interval, which will be sorted increasingly in the timetable. The following rules describe the behavior of the runtime dispatcher in case 0:

[0077] At the beginning of an execution interval, the respective job will be selected and launched to execute. If another job is being executed it will be preempted (i.e., interrupted).

[0078] At the end of said execution interval, if the job is still running, the job will be stopped and a new job will be selected as above.

[0079] If no execution interval exists for an interval of time (e.g., a gap in the timetable), the dispatcher will yield and wait until the next execution interval.

[0080] If a job finishes execution before the end of its execution interval, the dispatcher will yield.

[0081] According to case 1 (non-overlapping flexibility intervals) it may be provided that for jobs of the at least one first subset of task jobs the runtime dispatching means act according to the following rules:

[0082] if the current time is within the flexibility interval of a job, said job

[0083] will be launched to execute, if the core for said job is idle, i.e., no job is executing, and

[0084] if the core for said job is not idle, i.e., another job is executing, said selected job will be launched to execute at the instant of time that occurs first among the following:

[0085] instant of time at which the core for said job becomes idle, or

[0086] the current time reaches the start of the execution interval of the selected job.

[0087] The runtime dispatcher according to this case 1 will iterate over the timetable while keeping track of the progress of time in relation to the cycle. The dispatcher will compare the current time with the instants of time defining the start and end of each flexibility interval, which will be sorted increasingly in the timetable.

[0088] In the case of a not idle (processing) core, the job will in any event start at the release point of its execution interval. At the end of a flexibility interval, if the job is still running, the job will be stopped. If the instant corresponds to the start of the flexibility interval of another job, said job will be selected as described above.

[0089] If no flexibility interval exists for an interval of time (e.g., a gap in the timetable), the dispatcher (=runtime dispatching means) will yield until the next flexibility interval. Yields refers to giving away the control over the CPU or core since the dispatcher has nothing to do (there is no job that can be executed now).

[0090] If all jobs finish execution before the end of its flexibility interval, the dispatcher will yield until the next flexibility interval.

[0091] In general, the execution interval defines a guaranteed interval, but it is superseded by the flexibility interval to select and launch jobs for their execution. Therefore, the execution time of jobs may be larger than that of their execution intervals, which is an advantage in cases when the exact worst-case execution time of the task is not known, or it is underestimated for one or all jobs of said task, or those cases in which a job may eventually overrun.

[0092] Also in the general case, there is the possibility that another dispatcher is taking control and selecting other jobs of other tasks to execute in the "gaps" of the tt-schedule (i.e., when the tt-schedule "yields"). Therefore, when a job of a task in the tt-schedule is to be scheduled according to the dispatcher of the invention, there could be already a job of "another" task running, which can be either one in the sets of tasks related to the tt-schedule, or some other task related to some other dispatcher.

[0093] When such additional dispatchers are (for example hierarchically) active in the computer system, the execution of jobs of other tasks, for example fixed priority tasks, may occur within the flexibility interval of a selected job, up to, at most, the beginning of the execution interval. In such case, advantages of the invention are:

[0094] allowing jobs of other tasks, for example fixed priority tasks, to execute without jeopardizing the guaranteed execution interval of the tasks provided by the tt-schedule, and

[0095] allowing jobs of the tasks in the tt-schedule to execute as soon as the processing core remains idle within their flexibility interval, but not later than the instant in which the execution interval starts, and

[0096] allowing jobs of the task in the tt-schedule to execute until the end of its flexibility interval.

According to case 2 (strictly overlapping flexibility intervals) it may be provided that

[0097] for jobs of the at least one second subset of task jobs the runtime dispatching means act according to the following rules:

[0098] if the current time is within one or multiple flexibility intervals, a selection of one of the jobs for which said flexibility interval is equivalent will be performed based on one of the following defined criteria:

[0099] the job with the earliest execution interval will be selected and launched to execute, or

[0100] the job with the shortest execution time will be selected and launched to execute, or

[0101] the job resulting from a selection based on defined constraints, for example the earliest deadline, or the highest priority, will be selected and launched to execute (this allows the selection to be based on dynamic algorithms like Earliest Deadline First (EDF), or a fixed priority scheduler), or

[0102] the job for which the probability of finishing its execution in less time than that accounted for in its execution interval and providing more slack, i.e., unused CPU execution time, is higher will be selected.

[0103] The runtime dispatching means according to this case 2 will iterate over the timetable while keeping track of the progress of time in relation to the cycle. The dispatcher

will compare the current time with the instants of time defining the start and end of each flexibility interval, which will be sorted increasingly in the timetable. The selection in this sub-case is based on an estimate of the probability that a job will leave unused execution time. This time slack may be then used by other overrunning jobs within the subset of jobs sharing the flexibility interval bounds. The probability that a job finishes earlier can be dynamically measured during the execution over the past cycles, for example by calculating the moving average over the series of past execution cycles, or it can be based on static analysis and pre-defined before runtime.

[0104] In the context of case 2 it may be provided that the job selected and launched to execute will remain executing until any one of the following events occurs:

[0105] the job finishes its execution for the current cycle, or

[0106] the job reaches the amount of execution time corresponding to its execution interval, irrespective of the relative position of the current time within the flexibility interval during execution of the job, preferably plus any amount of slack available within the execution interval, calculated as:

[0107] the sum of existing idle intervals within the flexibility interval, calculated as the difference between the length of the flexibility interval and the sum of the execution intervals of all jobs within said flexibility interval (this can be statically determined prior to runtime; this condition guarantees that no job will be left with less than an interval corresponding to the length of its execution interval), or

[0108] the sum of slack provided by any job executing within the flexibility interval and not consumed by any other previous job (this can be dynamically determined based on the execution of jobs),

[0109] or

[0110] the job reaches the amount of execution time corresponding to its execution interval plus

[0111] a defined overrun budget, specified as

[0112] a fixed interval of time, e.g., 1 ms, or

[0113] a relative percentage of its own execution time, e.g., 10%,

or

[0114] the current time reaches the end of the flexibility interval.

[0115] The selection procedure will repeat as above with the remaining jobs (i.e., excluding the already executed job). If necessary, the slack account and statistical parameters will be updated.

[0116] Once all jobs have finished their execution for this cycle, the dispatcher will yield until the end of the flexibility interval, and then repeat.

[0117] According to the most generic case, case 3, the flexibility intervals of different jobs may partially overlap with flexibility or execution intervals of other jobs. Note that execution intervals of different jobs cannot overlap. The runtime dispatcher according to this case will iterate over the timetable while keeping track of the progress of time in relation to the cycle. The dispatcher will compare the current time with the instants of time defining the start and end of each flexibility interval and execution interval, which will be sorted increasingly in the timetable.

[0118] In this context it may be provided that for jobs of the at least one third subset of jobs the runtime dispatching means act according to the following rules:

[0119] when the current time reaches the start of a flexibility interval, the jobs related to said flexibility interval will be added to a list of eligible jobs, and wherein

[0120] when one of the following conditions is met:

[0121] the core on which the job is to be executed becomes idle,

[0122] a currently executing job finishes its execution, or is stopped, or is forced to stop,

[0123] the current time reaches the start of the execution interval of one of the jobs in the list of eligible jobs,

[0124] the runtime dispatching means will select the job from the list of eligible jobs for which the start of its execution interval occurs earlier.

[0125] Once a job selected from the list of eligible jobs, said job is removed from said list and launched for execution. When the list of eligible jobs is empty, the runtime dispatching means will yield.

[0126] It may further be provided that the execution of a job is stopped or the job is forced to stop execution

[0127] when the current time equals the completion point of the execution interval of said job and there is a job in the list of eligible jobs for which the start of its execution interval equals the current time, or

[0128] the current time is later than the completion point of the execution interval of said job and there is a job in the list of eligible jobs for which the start of its execution interval equals the current time, or

[0129] when the current time reaches the end of the flexibility interval of said executing job, if no job is in the list of eligible jobs for which the start of its execution interval equals the current time.

[0130] In this context is should be noted that it may be provided that a job finishes its execution by itself before the above conditions are met.

[0131] According to the invention, at least one of the cases 1-3 and the corresponding policy of the dispatching means is present. However, it may also be possible, that other cases, also case 0, or all cases 0, 1-3 are present in a schedule. The dispatching means are configured to deal with all those cases which are present, implementing at least one of the runtime dispatching policies described in cases 1, 2, and 3, in combination, or not, with the policy described in case 0.

[0132] Furthermore, the invention refers to runtime dispatching means for selecting jobs of tasks and launching said jobs to execution in a computer system, in particular for the use in a method as described above, according to a global time-triggered schedule,

[0133] wherein the computer system comprises one or more cores on one or more central processing units of the computer system,

[0134] wherein said global time-triggered schedule comprises one time-triggered schedule, "core schedule", for each of said cores, wherein the jobs of tasks of the set of tasks to be executed on a specific core are executed on said core according to the core schedule of said core,

[0135] wherein the global time-triggered schedule specifies a cycle with a cycle length for the set of tasks,

[0136] wherein the core schedule of a core specifies for the jobs of each task of the set of tasks, which is to be executed on said core:

[0137] an execution interval for each task instance, "job", of the task, during which execution interval the job can be executed, wherein each execution interval comprises a release point, at which the execution interval starts, and a completion point, at which the execution interval ends, wherein the duration of an execution interval is the duration of the time span between the release point and the completion point of the execution interval, wherein the execution interval of a job defines a time interval during which the job has a guarantee to be executed on the core, and

[0138] a period according to which period the jobs of said task are executed on the core,

[0139] wherein the execution interval of each job of the task is iterated cyclically according to the cycle, and

[0140] wherein the execution intervals of any two jobs of the same and of different tasks do not overlap, and

[0141] wherein the cycle is a common multiple of all periods of all tasks in the set of tasks.

[0142] The runtime dispatching means is characterized in that it is configured to select and launch jobs of tasks for execution according to the core schedule, taking into account all flexibility intervals of the core schedule.

[0143] The invention relates to runtime dispatching means, e.g., a runtime dispatcher module, for example the operating system dispatcher, responsible for the selection of one job among the jobs to be executed on the core, and launch said job to execution. Said selection is performed among the subset of eligible jobs, normally referred as the set of ready jobs, of the set of jobs, and relates to the respective intervals of each job depending on the above-mentioned cases.

[0144] In some cases, a runtime dispatcher may implement multiple dispatching policies, or in other cases multiple dispatchers may coexist simultaneously in a computer system. For example, a system may consist of two dispatchers, one implementing a fixed priority policy and one implementing a time-triggered policy, wherein each said dispatcher relates to a separate set of jobs.

[0145] Some computer systems, for example, computer systems based on GNU/Linux, implement several runtime dispatchers organized hierarchically. When the top dispatcher has no job to select for execution among the list of eligible jobs of its set of tasks, it yields and delegates to the next dispatchers in the hierarchy to perform a selection. If no dispatcher has eligible jobs to select, the respective core (CPU) idles until a job becomes eligible.

[0146] In connection with case 1 it may be provided that the runtime dispatching means is configured to act for first subsets of jobs, for which the core schedule specifies flexibility intervals which do not overlap with each other, wherein at least one of the flexibility intervals of a job is longer than the execution of said job according to the following rules:

[0147] if the current time is within the flexibility interval of a job, said job

[0148] will be launched to execute, if the core for said job is idle, i.e., no job is executing, and

[0149] if the core for said job is not idle, i.e., another job is executing, said selected job will be launched to execute at the instant of time that occurs first among the following:

[0150] instant of time at which the core for said job becomes idle, or

[0151] the current time reaches the start of the execution interval of the selected job.

[0152] In connection with case 2 it may be provided that the runtime dispatching means is configured to act for second subsets of jobs, for which the core schedule specifies flexibility intervals so that the flexibility intervals of two or more jobs of said second subset of jobs are identical, i.e., for said two or more jobs of said second subset the start instant of their flexibility intervals is the same instant and the end instant of their flexibility intervals is the same instant according to the following rules:

[0153] if the current time is within one or multiple flexibility intervals, a selection of one of the jobs for which said flexibility interval is equivalent will be performed based on one of the following defined criteria:

[0154] the job with the earliest execution interval will be selected and launched to execute, or

[0155] the job with the shortest execution time will be selected and launched to execute, or

[0156] the job resulting from a selection based on defined constraints, for example the earliest deadline, or the highest priority, will be selected and launched to execute, or

[0157] the job for which the probability of finishing its execution in less time than that accounted for in its execution interval and providing more slack, i.e., unused CPU execution time, is higher will be selected.

[0158] In connection with case 2 it may further be provided that the runtime dispatching means is configured such that the job selected and launched to execute will remain executing until any one of the following events occurs:

[0159] the job finishes its execution for the current cycle, or

[0160] the job reaches the amount of execution time corresponding to its execution interval, irrespective of the relative position of the current time within the flexibility interval during execution of the job, preferably plus any amount of slack available within the execution interval, calculated as:

[0161] the sum of existing idle intervals within the flexibility interval, calculated as the difference between the length of the flexibility interval and the sum of the execution intervals of all jobs within said flexibility interval, or

[0162] the sum of slack provided by any job executing within the flexibility interval and not consumed by any other previous job,

[0163] or

[0164] the job reaches the amount of execution time corresponding to its execution interval plus a defined overrun budget, specified as

[0165] a fixed interval of time, e.g., 1 ms, or

[0166] a relative percentage of its own execution time, e.g., 10%,

or

[0167] the current time reaches the end of the flexibility interval.

[0168] In connection with case 3 it may be provided that the runtime dispatching means is configured to act for third subsets of jobs, for which the core schedule specifies that the flexibility intervals of any two jobs of said third subset partially overlap according to the following rules:

[0169] when the current time reaches the start of a flexibility interval, the jobs related to said flexibility interval will be added to a list of eligible jobs, and wherein

[0170] when one of the following conditions is met:

[0171] the core on which the job is to be executed becomes idle,

[0172] a currently executing job finishes its execution, or is stopped, or is forced to stop,

[0173] the current time reaches the start of the execution interval of one of the jobs in the list of eligible jobs,

[0174] the runtime dispatching means will select the job from the list of eligible jobs for which the start of its execution interval occurs earlier.

[0175] In connection with case 3 it may further be provided that the execution of a job is stopped or the job is forced to stop execution:

[0176] when the current time equals the completion point of the execution interval of said job and there is a job in the list of eligible jobs for which the start of its execution interval equals the current time, or

[0177] the current time is later than the completion point of the execution interval of said job and there is a job in the list of eligible jobs for which the start of its execution interval equals the current time, or

[0178] when the current time reaches the end of the flexibility interval of said executing job, if no job is in the list of eligible jobs for which the start of its execution interval equals the current time.

[0179] The invention also refers to a computer system comprising one or more cores on one or more central processing units, wherein the computer system is configured to execute tasks according to a method as described above, the computer system comprising runtime dispatching means as described above.

[0180] Furthermore, the invention relates to a method to generate a core schedule for the execution of jobs of tasks on a core of a computer system, in particular for the use in a method as described above, wherein the core schedule comprises flexibility intervals for the jobs to be executed on said core, the method comprising the steps of:

[0181] providing a time-triggered schedule for the tasks to be executed on said core, which time-triggered schedule comprises an execution interval for each job of each task and a period for each task, according to which period the jobs of said task are cyclically executed,

[0182] determine schedulability bounds based on said time-triggered schedule and based on a set of scheduling constraints, resulting in upper and lower bounds for each execution interval,

[0183] for example by iteratively increasing, starting with the execution interval of a first job of a task from said time-triggered schedule for said core, the execution interval while checking that the scheduling constraints are fulfilled, and

[0184] once in an iteration at least one scheduling constraint is not fulfilled, determine the extended execution interval of the previous iteration as the flexibility interval for said first job,

[0185] selected as first job until all jobs have been provided a flexibility interval,

[0186] selecting a next job and repeating the method until all jobs have been provided with a flexibility interval.

BRIEF DESCRIPTION OF THE DRAWINGS

[0187] The invention is described in a non-limiting manner with reference to the figures:

[0188] FIG. **1** depicts a time triggered schedule for a set TS_1 of jobs with no flexibility intervals.

[0189] FIG. **2** depicts a time-triggered schedule with flexibility intervals for a set TS_2 of jobs with non-overlapping flexibility intervals.

[0190] FIG. **3** depicts a time-triggered schedule with flexibility intervals for a set TS_3 of jobs with completely overlapping flexibility intervals.

[0191] FIG. **4** depicts a time-triggered schedule with flexibility intervals for a set of jobs with partially overlapping flexibility intervals.

[0192] FIGS. **5-8** depict exemplary execution instances, according to embodiments of the invention, of the time-triggered schedules depicted in FIGS. **1-4**.

DETAILED DESCRIPTION OF EMBODIMENTS

[0193] A computer system, for example an automotive computer system, comprises one or more hosts, in particular processors, wherein each host, h_i, in the set of said hosts, $H=\{h_1 \ldots h_n\}$, comprises one or more CPU cores, characterized by the set of CPU cores, $C^H=\{c^h{}_1 \ldots c^h{}_k\}$, wherein said CPU cores are configured to execute instructions. Software applications, for example automotive functions, comprise software components (tasks), which comprise a sequence of instructions that can be executed in said one or multiple CPU cores.

[0194] Software tasks are cyclically executed in said CPU cores, wherein each software task, sw_i, in the set of defined software tasks, $SW=\{sw_1 \ldots sw_m\}$, is characterized by

[0195] the length of its execution cycle, namely its period, or minimum interarrival time, p_i, and

[0196] its maximum required execution budget, w_i corresponding to the accrued time required in one CPU core to execute said task within one cycle, for example its worst-case computation time (WCET), or and estimation of the required computation time.

[0197] A software task, sw_i, may further be characterized by

[0198] its earliest release time, e_i, indicating the earliest relative time when the software task is ready for execution, relative to the beginning of a cycle, and/or

[0199] its execution deadline, d_i, indicating the latest relative point in time after the beginning of a cycle by which the software task shall complete the execution corresponding to said cycle.

[0200] The software tasks in the CPU are executed following a time-triggered scheduling paradigm, whereby a run-time component of said computer system (the dispatcher), for example the runtime dispatcher component of the operating system, selects and dispatches instances of

said tasks (jobs) to be executed for defined time intervals, or intervals, in said CPU cores following a predefined time-triggered schedule (the tt-schedule).

[0201] Each said interval is characterized by a start time, or time offset, o, and a duration, or length, b.

[0202] A tt-schedule for a CPU core, c, the "core schedule", relates to a timetable, TT^c, characterized by the length of the table cycle, ze, or schedule cycle, and a sequence of one or more elements, wherein each said element of said timetable is characterized by a reference to an assigned job of a software task, sw, from the set of software tasks, SW.

[0203] The periodic execution of tasks within the schedule cycle comprises one or more executions of task instances, or jobs, of said task represented by one said element in the timetable, wherein each job corresponds to an execution of the task according to its periodicity within the schedule cycle. If the task period is equal to the schedule cycle, there is one single job of said task within the schedule cycle, whereas if the schedule cycle is larger than the period, there are multiple jobs of said task according to the relation between the period and the schedule cycle. Note that the length of the schedule cycle, shall be a common multiple of the period of all tasks, for example, the least common multiple.

[0204] In the examples shown below, it is assumed for the sake of simplicity that each specific task is only executed once within a cycle with cycle length T (exactly spoken only one job of said task is executed within a cycle), so that in the following description for the sake of simplicity the wording "execution of a task" instead of the wording "execution of a job of said" is used.

[0205] According to the invention, each element of said timetable is further characterized by an additional flexibility interval, fi.

[0206] Table 1 shows an exemplary set of tasks (also denoted as "task set") TS_0, comprising three tasks (one job per task), sw_1, sw_2, and sw_3, characterized by their period p, early release time e (earliest start point), deadline d (latest completion point), and execution budget w.

TABLE 1

	sw_1	sw_2	sw_3
p	T	T	T
e	t_0	t_2	t_5
d	t_2	t_5	t_7
w	w_1	w_2	w_3

[0207] Reference sign "T" denotes the cycle length specified by the global time-triggered schedule.

[0208] FIG. **1** depicts an example of a time triggered schedule according to the state of the art, computed by an offline time-triggered scheduler for said task set TS_0. As can be seen from FIG. **1**, a fixed execution interval is assigned to each task and no flexibility intervals are provided, as is the case in the prior art.

[0209] Table 2 characterizes the tt-schedule from FIG. **1** in the form of a timetable, wherein the schedule cycle z is equal to the period p, (i.e., z=T) and the execution intervals are s_1, s_2, and s_3 (o . . . start time, b . . . duration/length),

TABLE 2

	o	b	sw
s_1	t_1	$t_2 - t_1$	sw_1
s_2	t_3	$t_4 - t_3$	sw_2
s_3	t_5	$t_6 - t_5$	sw_3

[0210] Table 3 shows an exemplary task set ("first" set of tasks), TS_1, comprising three tasks, sw_4, sw_5, and sw_6, characterized by their period p, early release time e, deadline d, and execution budget w.

TABLE 3

	sw_4	sw_5	sw_6
p	T	T	T
e	t_0	t_2	t_5
d	t_2	t_5	t_7
w	w_4	w_5	w_6

[0211] FIG. **2** depicts a time-triggered schedule for the task set TS_1 from table 3, with execution intervals s_4, s_5, s_6 and with non-overlapping flexibility intervals fi_4, fi_5, and fi_6, for the tasks of task set TS_1.

[0212] Table 4 characterizes the tt-schedule from FIG. **2** in the form of a timetable, wherein the schedule cycle is equal to the period T, (i.e., z=T) and the execution intervals are s_4, s_5, and s_6. The parameters o and b refer to the execution interval of the respective task, fi denotes the flexibility interval of the respective task.

TABLE 4

	o	b	sw	fi
s_4	t_1	$t_2 - t_1$	sw_4	$fi_4 = (t_0, t_2)$
s_5	t_3	$t_4 - t_3$	sw_5	$fi_5 = (t_2, t_5)$
s_6	t_5	$t_6 - t_5$	sw_6	$fi_6 = (t_5, t_7)$

[0213] Table 5 shows an exemplary task set ("second" set of tasks), TS_2, comprising three tasks, sw_7, sw_8, and sw_9, characterized by their period p, early release time e, deadline d, and execution budget w.

TABLE 5

	sw_7	sw_8	sw_9
p	T	T	T
e	t_0	t_0	t_0
d	t_7	t_7	t_7
w	w_7	w_8	w_9

[0214] FIG. **3** depicts a time-triggered schedule with flexibility intervals for a task set with identical, completely overlapping flexibility intervals, fi_7, fi_5, and fi_9, for the task set TS_2.

[0215] Table 6 characterizes the tt-schedule from FIG. **3** in the form of a timetable, wherein the schedule cycle is equal to the period T, (i.e., z=T) and the execution intervals (parameters o, b) are s_7, s_8, and s_9.

TABLE 6

	o	b	sw	Fi
s_7	t_1	$t_2 - t_1$	sw_7	$fi_7 = (t_0, t_7)$
s_8	t_3	$t_4 - t_3$	sw_8	$fi_8 = (t_0, t_7)$
s_9	t_5	$t_6 - t_5$	sw_9	$fi_9 = (t_0, t_7)$

[0216] An additional exemplary set of tasks ("third" set of tasks), TS_3, comprising three tasks, sw_{10}, sw_{11}, and sw_{12}, is characterized in Table 7, by the period, p, early release time, e, deadline, d, and execution budget, w, of each task sw_{10}, sw_{11}, sw_{12} of the task set TS_3.

TABLE 7

	sw_{10}	sw_{11}	sw_{12}
p	T	T	T
e	t_0	t_0	t_0
d	t_7	t_7	t_7
w	w_{10}	w_{11}	w_{12}

[0217] FIG. **4** depicts a time-triggered schedule with flexibility intervals for a task set with partially overlapping flexibility intervals, fi_{10}, fi_{11}, and fi_{12}, for the task set TS_3 from Table 7.

[0218] Table 8 characterizes the tt-schedule from FIG. **4** in the form of a timetable, wherein the schedule cycle is equal to the period T, (i.e., z=T) and the execution intervals (parameters o, b) are s_{10}, s_{11}, s_{12}.

TABLE 8

	o	b	sw	fi
s_{10}	t_1	$t_3 - t_1$	sw_{10}	$fi_{10} = (t_0, t_3)$
s_{11}	t_4	$t_5 - t_4$	sw_{11}	$fi_{11} = (t_2, t_6)$
s_{12}	t_6	$t_7 - t_6$	sw_{12}	$fi_{12} = (t_3, t_7)$

[0219] FIGS. **5-8** depict exemplary execution instances ("jobs") of the time-triggered schedules depicted in FIGS. **1-4** and characterized in the tables described above.

Case 0: (State of the Art)

[0220] The exemplary task set, TS_0, comprises three tasks, sw_1, sw_2, and sw_3, as described above.

[0221] A time-triggered scheduler accordingly computes a time-trigger schedule for TS_0 as shown in Table 2, with no flexibility interval.

[0222] FIG. **5** depicts a possible execution trace of said task set, TS_0, for the purpose of this example, wherein the task execution time for each task in the task set is less than or equal to the corresponding execution interval of the task.

[0223] The starting execution time for each task is strictly bound to the start of the execution interval determined by the time-triggered schedule, regardless of the early termination of any preceding task in the corresponding time-triggered schedule. For example, the execution of sw_2 in FIG. **5** starts at time t_3 independent of the fact the task sw_1 has already finished execution at an earlier point in time, t_{11}, with respect to the end of the execution interval, t_2, of task sw_1 as defined in the time-triggered schedule Table 2.

[0224] Also, according to a general time-triggered dispatcher without the additional advantages of the present invention, the end of the execution time of task sw_3 in FIG. **5** is upper bounded to the end of the execution interval, to, regardless of whether the task is already finished with its execution at this point t_6 or not.

Case 1:

[0225] A first set of tasks, TS_1, comprising three tasks, sw_4, sw_5, and sw_6, is described above in detail and characterized in Table 3. A time-triggered scheduler including advantages of the invention may compute a time-trigger schedule as characterized in Table 4 and depicted in FIG. **2**, with flexibility intervals for the tasks that do not overlap.

[0226] FIG. **6** depicts a possible execution trace of said first task set, TS_1, for the purpose of this example, wherein the task available execution time for some tasks in the task set is larger than the corresponding task execution interval.

[0227] A time-triggered dispatcher (dispatching means) according to the invention may select and launch tasks for execution as depicted in FIG. **6**. For example, in the exemplary trace the execution of sw_4 in FIG. **6** may start (due to its flexibility interval) at time to, ahead of the start of the start of its execution interval, t_1, if the CPU core is idle at that time.

[0228] Task sw_5 starts in this example during its flexibility interval at t**31**, later than the starting instance t_3 of its execution interval s_5 ($=t_4-t_3$) and ends later than the end t_4 of its execution interval (but before the end of its flexibility interval). Accordingly, the start of the execution time of task sw_5 according to FIG. **6**, t_{31}, is bounded to the start of neither its flexibility interval, t_2, nor its execution interval, t_4, for example allowing the CPU core to execute other tasks, not depicted in the figure, or remain in a low power mode to save energy.

[0229] The end of the execution time of task sw_6 in FIG. **6** is upper bounded to the end of the flexibility interval, t_7, regardless of the end of the defined end of the execution interval, to, for example allowing said task sw_6 to execute longer than its assigned execution interval if required.

Case 2:

[0230] Yet another exemplary task set, second task set TS_2, comprising three tasks, sw_7, sw_8, and sw_9, is characterized in Table 5, as described in detail above. A time-triggered scheduler including advantages of this invention may compute a time-trigger schedule as characterized in Table 6 with flexibility intervals for the tasks, as depicted in FIG. **3**. In this example the flexibility intervals are equal.

[0231] FIG. **7** depicts a possible execution trace of said task set, for the purpose of this example, wherein the task execution time for some tasks in the task set is larger than the corresponding task execution interval.

[0232] According to a time-triggered dispatcher according to the invention, the starting execution time for each task are bounded by the corresponding flexibility interval determined by the time-triggered schedule, wherein said flexibility interval is equal for all tasks in the task set. For example, in the exemplary trace the execution of sw_9 in FIG. **7** starts at time to, ahead of the start of its execution interval, t_5, and prior to tasks with an earlier execution interval, sw_7 and sw_8, for example due to the task sw_9 having a shorter execution time than its budgeted execution interval s_9, which for example is identified by historical observation of preceding cycles, or due to the dispatching of task based on statically

defined, or dynamically assigned, priorities, whereby sw_9 is assigned a higher priority than sw_8 and sw_7.

[0233] Furthermore, the execution time (t_7–t_3) of task sw_5 in the example according to FIG. **7** is larger than the execution budget corresponding to its execution interval, t_4–t_3, allowing a larger execution time for sw_5 which for example includes the accrued slack corresponding to unused time from preceding tasks plus the additional time within the flexibility interval, for example allowing sw_5 to implement a progressive algorithm maximizing its execution time until the end of the flexibility interval, whereby the functional performance of said algorithm is improved proportionally to its runtime.

Case 3:

[0234] A further exemplary task, TS_3, the third task set, comprising three tasks, sw_{10}, sw_{11}, and sw_{12}, is characterized in Table 7 as described above in detail. A time-triggered scheduler may compute a time-triggered schedule as described in Table 8 by the beginning of the execution interval, o, the length of the execution interval, b, the related task, sw, and the flexibility interval, fi, of each execution interval, and depicted in FIG. **4**. Said time-trigger schedule for the third task set is provided with flexibility intervals computed according to embodiments of this invention and wherein said flexibility intervals partially overlap.

[0235] FIG. **8** depicts a possible execution trace of said third task set, for the purpose of this example, wherein the task execution time for some tasks in the task set is larger than the corresponding task execution interval.

[0236] According to a time-triggered dispatcher according to the invention, the starting execution time for each task are bounded by the corresponding flexibility interval determined by the time-triggered schedule, which flexibility intervals are partially overlapping. For example, in the exemplary trace the execution of sw_{10} in FIG. **8**, may start at time to, ahead of the start of its execution interval, t_2. Similarly, task sw_{11}, may start at time t_{21}, right after sw_{10} finishes execution and ahead of the start of its execution interval, t_4.

[0237] Furthermore, the execution time (t_{61}–t_{41}) of task sw_{12} in FIG. **8** is larger than the execution budget corresponding to its execution interval, t_7-t_6, allowing a larger execution time for sw_{12}.

REFERENCES

[0238] [1] S. S. Craciunas, R. Serna Oliver—Combined Task- and Network-level Scheduling for Distributed Time-triggered Networked Systems. Real-Time Systems, Volume 52, Issue 2, pp. 161-200, Springer, 2016.

[0239] [2] Anna Minaeva and Zdeněk Hanzálek. 2021. Survey on Periodic Scheduling for Time-triggered Hard Real-time Systems. ACM Comput. Surv. 54, 1, Article 23 (January 2022), 32 pages. https://doi.org/10.1145/3431232

[0240] [3] Richard Hladik, Anna Minaeva, and Zdeněk Hanzálek. 2020. On the Complexity of a Periodic Scheduling Problem with Precedence Relations. In Combinatorial Optimization and Applications: 14th International Conference, COCOA 2020, Dallas, TX, USA, Dec. 11-13, 2020, Proceedings. Springer-Verlag, Berlin, Heidelberg, 107-124. https://doi.org/10.1007/978-3-030-64843-5_8

[0241] [4] McLean S D, Juul Hansen E A, Pop P and Craciunas S S (2022) Configuring ADAS Platforms for Automotive Applications Using Metaheuristics. Front. Robot. AI 8:762227. doi: 10.3389/frobt.2021.762227

[0242] [5] A. F. Mills and J. H. Anderson, "A Multiprocessor Server-Based Scheduler for Soft Real-Time Tasks with Stochastic Execution Demand," 2011 IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications, Toyama, Japan, 2011, pp. 207-217, doi: 10.1109/RTCSA.2011.30.

[0243] [6] Ramon Serna Oliver, Paraskevas Karachatzis, and Silviu Craciunas. "Method to execute a mode-transition in a multi-mode computer system." U.S. patent application Ser. No. 17/930,811.

[0244] [7] Baruah, S. K., Rosier, L. E. & Howell, R. R. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. Real-Time Syst 2, 301-324 (1990). https://doi.org/10.1007/BF01995675

[0245] [8] R. Pellizzoni and G. Lipari, "Feasibility analysis of real-time periodic tasks with offsets," Real-Time Syst., vol. 30, no. 1-2, pp. 105-128, 2005

[0246] [9] Kong, W.; Nabi, M.; Goossens, K. Run-Time Recovery and Failure Analysis of Time-Triggered Traffic in Time Sensitive Networks. IEEE Access 2021, 9, 91710-91722.

[0247] [10] Li, J.; Xiong, H.; Li, Q.; Xiong, F.; Feng, J. Run-Time Reconfiguration Strategy and Implementation of Time-Triggered Networks. Electronics 2022, 11, 1477. https://doi.org/10.3390/electronics11091477

That which is claimed is:

1. A method to execute jobs of a set of tasks in a computer system according to a global time-triggered schedule,

wherein the computer system comprises one or more cores on one or more central processing units of the computer system,

wherein said global time-triggered schedule comprises one time-triggered schedule, "core schedule", for each of said cores, wherein the tasks of the set of tasks to be executed on a specific core are executed on said core according to the core schedule of said core,

wherein the global time-triggered schedule specifies a cycle with a cycle length for the set of tasks,

wherein the core schedule of a core specifies for each task of the set of tasks, which is to be executed on said core:

an execution interval for each task instance, "job", of the task, during which execution interval the job can be executed, wherein each execution interval comprises a release point, at which the execution interval starts, and a completion point, at which the execution interval ends, wherein the duration of an execution interval is the duration of the time span between the release point and the completion point of the execution interval, wherein the execution interval of a job defines a time interval during which the job has a guarantee to be executed on the core, and

a period according to which period the jobs of said task are executed on the core,

wherein the execution interval of each job of the task is iterated cyclically according to the cycle, and

wherein the execution intervals of any two jobs of the same and of different tasks do not overlap, and

wherein the cycle is a common multiple of all periods of all tasks in the set of tasks,

wherein:

the core schedule specifies for each job a flexibility interval, wherein the duration of the flexibility interval of a job is equal or longer than the execution interval of said job,

at least the flexibility interval of one job is longer than the execution interval of said job,

the flexibility interval of a job starts no later than the release point of the execution interval of said job, and the flexibility interval of a job ends no sooner than the completion point of the execution interval of said job,

the flexibility of a job defines a time interval during which said job can be executed, and

runtime dispatching means are provided, which select and launch jobs for execution according to the global time-triggered schedule, taking into account for each core all flexibility intervals as specified in the core schedule.

2. The method according to claim **1**, wherein among the jobs of those tasks of the set of tasks that are to be executed on the core at least one first subset of jobs is provided, wherein for the jobs of said at least one first subset the core schedule specifies flexibility intervals which do not overlap with each other, wherein at least one of the flexibility intervals of a job is longer than the execution of said job.

3. The method according to claim **1**, wherein among those tasks of the set of tasks that are to be executed on the core at least one second subset of jobs is provided, wherein for the jobs of said at least one second subset the core schedule specifies flexibility intervals so that the flexibility intervals of two or more jobs of said at least one second subset of jobs are identical, i.e., for said two or more jobs of said at least one second subset the start instant of their flexibility intervals is the same instant and the end instant of their flexibility intervals is the same instant.

4. The method according to claim **1**, wherein among those tasks of the set of tasks that are to be executed on the core at least one third subset of jobs is provided, wherein the flexibility intervals of any two jobs of said at least one third subset partially overlap.

5. The method according to claim **2**, wherein for jobs of the at least one first subset of task jobs the runtime dispatching means act according to the following rules:

if the current time is within the flexibility interval of a job, said job

will be launched to execute, if the core for said job is idle, i.e., no job is executing, and

if the core for said job is not idle, i.e., another job is executing, said selected job will be launched to execute at the instant of time that occurs first among the following:

instant of time at which the core for said job becomes idle, or

the current time reaches the start of the execution interval of the selected job.

6. The method according to claim **3**, wherein for jobs of the at least one second subset of task jobs the runtime dispatching means act according to the following rules:

if the current time is within one or multiple flexibility intervals, a selection of one of the jobs for which said flexibility interval is equivalent will be performed based on one of the following defined criteria:

the job with the earliest execution interval will be selected and launched to execute, or

the job with the shortest execution time will be selected and launched to execute, or

the job resulting from a selection based on defined constraints, for example the earliest deadline, or the highest priority, will be selected and launched to execute, or

the job for which the probability of finishing its execution in less time than that accounted for in its execution interval and providing more slack, i.e., unused CPU execution time, is higher will be selected.

7. The method according to claim **6**, wherein the job selected and launched to execute will remain executing until any one of the following events occurs:

the job finishes its execution for the current cycle, or

the job reaches the amount of execution time corresponding to its execution interval, irrespective of the relative position of the current time within the flexibility interval during execution of the job, preferably plus any amount of slack available within the execution interval, calculated as:

the sum of existing idle intervals within the flexibility interval, calculated as the difference between the length of the flexibility interval and the sum of the execution intervals of all jobs within said flexibility interval, or

the sum of slack provided by any job executing within the flexibility interval and not consumed by any other previous job,

or

the job reaches the amount of execution time corresponding to its execution interval plus a defined overrun budget, specified as

a fixed interval of time, e.g., 1 ms, or

a relative percentage of its own execution time, e.g., 10%, or

the current time reaches the end of the flexibility interval.

8. The method according to claim **4**, wherein for jobs of the at least one third subset of jobs the runtime dispatching means act according to the following rules:

when the current time reaches the start of a flexibility interval, the jobs related to said flexibility interval will be added to a list of eligible jobs, and wherein

when one of the following conditions is met:

the core on which the job is to be executed becomes idle,

a currently executing job finishes its execution, or is stopped, or is forced to stop,

the current time reaches the start of the execution interval of one of the jobs in the list of eligible jobs,

the runtime dispatching means will select the job from the list of eligible jobs for which the start of its execution interval occurs earlier.

9. The method according to claim **8**, wherein the execution of a job is stopped or the job is forced to stop execution

when the current time equals the completion point (completion instant) of the execution interval of said job and there is a job in the list of eligible jobs for which the start of its execution interval equals the current time, or

the current time is later than the completion point of the execution interval of said job and there is a job in the

list of eligible jobs for which the start point (start instant) of its execution interval equals the current time, or

when the current time reaches the end of the flexibility interval of said executing job, if no job is in the list of eligible jobs for which the start of its execution interval equals the current time.

10. A runtime dispatching means for selecting jobs of tasks and launching said jobs to execution in a computer system, for the use in a method according to claim **1**, according to a global time-triggered schedule,

wherein the computer system comprises one or more cores on one or more central processing units of the computer system,

wherein said global time-triggered schedule comprises one time-triggered schedule, "core schedule", for each of said cores, wherein the jobs of tasks of the set of tasks to be executed on a specific core are executed on said core according to the core schedule of said core,

wherein the global time-triggered schedule specifies a cycle with a cycle length for the set of tasks,

wherein the core schedule of a core specifies for the jobs of each task of the set of tasks, which is to be executed on said core:

an execution interval for each task instance, "job", of the task, during which execution interval the job can be executed, wherein each execution interval comprises a release point, at which the execution interval starts, and a completion point, at which the execution interval ends, wherein the duration of an execution interval is the duration of the time span between the release point and the completion point of the execution interval, wherein the execution interval of a job defines a time interval during which the job has a guarantee to be executed on the core, and

a period according to which period the jobs of said task are executed on the core,

wherein the execution interval of each job of the task is iterated cyclically according to the cycle, and

wherein the execution intervals of any two jobs of the same and of different tasks do not overlap,

wherein the cycle is a common multiple of all periods of all tasks in the set of tasks, and

wherein it is configured to select and launch jobs of tasks for execution according to the core schedule, taking into account all flexibility intervals of the core schedule.

11. The runtime dispatching means according to claim **10**, wherein the runtime dispatching means is configured to act for first subsets of jobs, for which the core schedule specifies flexibility intervals which do not overlap with each other, wherein at least one of the flexibility intervals of a job is longer than the execution of said job according to the following rules:

if the current time is within the flexibility interval of a job, said job

will be launched to execute, if the core for said job is idle, i.e., no job is executing, and

if the core for said job is not idle, i.e., another job is executing, said selected job will be launched to execute at the instant of time that occurs first among the following:

instant of time at which the core for said job becomes idle, or

the current time reaches the start of the execution interval of the selected job.

12. The runtime dispatching means according to claim **10**, wherein the runtime dispatching means is configured to act for second subsets of jobs, for which the core schedule specifies flexibility intervals so that the flexibility intervals of two or more jobs of said second subset of jobs are identical, i.e., for said two or more jobs of said second subset the start instant of their flexibility intervals is the same instant and the end instant of their flexibility intervals is the same instant according to the following rules:

if the current time is within one or multiple flexibility intervals, a selection of one of the jobs for which said flexibility interval is equivalent will be performed based on one of the following defined criteria:

the job with the earliest execution interval will be selected and launched to execute, or

the job with the shortest execution time will be selected and launched to execute, or

the job resulting from a selection based on defined constraints, for example the earliest deadline, or the highest priority, will be selected and launched to execute, or

the job for which the probability of finishing its execution in less time than that accounted for in its execution interval and providing more slack, i.e., unused CPU execution time, is higher will be selected.

13. The runtime dispatching means according to claim **12**, wherein the runtime dispatching means is configured such that the job selected and launched to execute will remain executing until any one of the following events occurs:

the job finishes its execution for the current cycle, or

the job reaches the amount of execution time corresponding to its execution interval, irrespective of the relative position of the current time within the flexibility interval during execution of the job, preferably plus any amount of slack available within the execution interval, calculated as:

the sum of existing idle intervals within the flexibility interval, calculated as the difference between the length of the flexibility interval and the sum of the execution intervals of all jobs within said flexibility interval, or

the sum of slack provided by any job executing within the flexibility interval and not consumed by any other previous job,

or

the job reaches the amount of execution time corresponding to its execution interval plus a defined overrun budget, specified as

a fixed interval of time, e.g., 1 ms, or

a relative percentage of its own execution time, e.g., 10%,

or

the current time reaches the end of the flexibility interval.

14. The runtime dispatching means according to claim **10**, wherein the runtime dispatching means is configured to act for third subsets of jobs, for which the core schedule specifies that the flexibility intervals of any two jobs of said third subset partially overlap according to the following rules:

when the current time reaches the start of a flexibility interval, the jobs related to said flexibility interval will be added to a list of eligible jobs, and wherein

when one of the following conditions is met:

the core on which the job is to be executed becomes idle,

a currently executing job finishes its execution, or is stopped, or is forced to stop,

the current time reaches the start of the execution interval of one of the jobs in the list of eligible jobs,

the runtime dispatching means will select the job from the list of eligible jobs for which the start of its execution interval occurs earlier.

15. The runtime dispatching means according to claim **14**, wherein the execution of a job is stopped or the job is forced to stop execution:

when the current time equals the completion point of the execution interval of said job and there is a job in the list of eligible jobs for which the start of its execution interval equals the current time, or

the current time is later than the completion point of the execution interval of said job and there is a job in the list of eligible jobs for which the start of its execution interval equals the current time, or

when the current time reaches the end of the flexibility interval of said executing job, if no job is in the list of eligible jobs for which the start of its execution interval equals the current time.

16. A computer system comprising one or more cores on one or more central processing units, wherein the computer system is configured to execute tasks according to a method according to claim **1**.

17. A method to generate a core schedule for the execution of jobs of tasks on a core of a computer system, for the use in a method according to claim **1**, wherein the core schedule comprises flexibility intervals for the jobs to be executed on said core, the method comprising the steps of:

providing a time-triggered schedule for the tasks to be executed on said core, which time-triggered schedule comprises an execution interval for each job of each task and a period for each task, according to which period the jobs of said task are cyclically executed,

determine schedulability bounds based on said time-triggered schedule and based on a set of scheduling constraints, resulting in upper and lower bounds for each execution interval,

for example by iteratively increasing, starting with the execution interval of a first job of a task from said time-triggered schedule for said core, the execution interval while checking that the scheduling constraints are fulfilled, and

once in an iteration at least one scheduling constraint is not fulfilled, determine the extended execution interval of the previous iteration as the flexibility interval for said first job,

selected as first job until all jobs have been provided a flexibility interval,

selecting a next job and repeating the method until all jobs have been provided with a flexibility interval.

* * * * *