Semi-partitioned Model for Dual-core Mixed Criticality System

Hao Xu
Department of Computer Science
University of York, UK
hx569@york.ac.uk

Alan Burns
Department of Computer Science
University of York, UK
alan.burns@york.ac.uk

ABSTRACT

Many Mixed Criticality Algorithms have been developed with an assumption that lower criticality level tasks may be abandoned to guarantee the schedulability of higher criticality tasks when the criticality level of the system changes. But it is valuable to explore a way that all of the tasks remain schedulable throughout the criticality level changes. This paper introduces a possible semi-partitioned model which allows all of the tasks to remain schedulable if only one core increases its criticality level. In such a model, some lower criticality tasks are allowed to migrate instead of being abandoned and detailed response time analysis of the model is given. This paper also addresses possible task allocation approaches, priority assignment and choice of migration tasks. An evaluation is made upon different semi-partitioned approaches and recommendations are given.

1. INTRODUCTION

A system containing tasks with different criticality levels is called a Mixed Criticality System (MCS). In such a system, each task may have different Worst Case Execution Time (WCET) estimates for its different criticality levels, and normally the WCET of a task will increase if its criticality level increases. Vestal [16] firstly introduces an algorithm allowing all of the tasks with different criticality levels to remain schedulable regardless of the changes of the system mode. System mode change refers to a switch of defined operating modes of the system, which is generally controlled by a mode change protocol [7]. In MCS, a system mode change mostly refers to a change of the criticality level of the system, for example, the system rises from a lower criticality level to a higher criticality level. Based on this, a variety of algorithms, such as AMC [5] and EDF-VD [4], were developed to improve the scheduling efficiency of MCS. Most of these algorithms were invented under the assumption that there are two criticality levels, LO-crit level and HI-crit level (HI-crit level is higher than LO-crit level), and LO-crit tasks may be terminated in order to ensure the execution of HI-crit tasks when the criticality level of the system increases. Although these LO-crit tasks will be brought back to execution after the system level restores, it would be better if these tasks can remain executing throughout the system mode change. But it is not possible on single core platforms as the maximum computation capability of a core is fixed and it is often too expensive to increase the performance of the core. Thus, many researches [2][12][15] have been carried out to address MCS on multi-core platform, as one of the key features of a multi-core platform is that tasks may be able to migrate from one core to others, which provides more flexibility for scheduling. This migration progress is supported by many operation systems, such as Linux. In Linux, an attribute called CPU affinity is used to "bond" a task to a given set of CPUs [13]. In other words, the Linux scheduler will assign tasks to certain cores according to their CPU affinity value. Considering that, a migration progress may be simplified to changes of the CPU affinity attribute of a task.

Multi-core scheduling algorithms can generally be divided into three categories [9]: partitioned scheduling, global scheduling and semi-partitioned scheduling. Partitioned scheduling, where tasks are statically mapped to processors, provides a stable and predictable implementation which is preferable for safety critical applications, while global scheduling allows tasks to migrate from one processor to others during execution which potentially provides higher overall utilisation. Semi-partitioned scheduling is a mixture of the previous two algorithms in that hard real-time tasks may be statically mapped to processors and other tasks are able to migrate for flexibility. Referring to MCS, HI-crit tasks may be statically partitioned on processors in order to guarantee their execution, while some LO-crit tasks may be migrated when a system mode change is detected on their executing core.

This paper explores semi-partitioned scheduling on multi-core MCS in consideration that all tasks remain schedulable if only one core enters the HI-crit mode. Firstly, it gives some background information, including useful notations, the AMC algorithm and approaches to task allocations. Then it describes the semi-partitioned model based on views of states; following with detailed response time analysis of the model and an example. This paper also addresses possible approaches to task allocation, priority assignment and choice of migration tasks with reference to the semi-partitioned model. After that, an experiment to test the performance of the model comparing with a none-migration model is introduced and the experiment results are discussed. Conclusion are made at the end.

RTNS 2015, November 04 - 06, 2015, Lille, France.

http://dx.doi.org/10.1145/2834848.2834865.

2. RELATED WORKS, MODELS AND NOTATION

2.1 MCS Notation

Since tasks may have different WCET for their different criticality levels, the notation for MCS is a bit different from the standardised notation for real time system. Table 1 shows the symbols used in this paper.

Notation	Description
τ_i	Task i
D_i	The deadline of task τ_i
T_i	The period of task τ_i
L_i	The criticality level of task τ_i
$C_i(L_i)$	The WCET of task τ_i at criticality level L_i
$U_i(L_i)$	The utilisation of task τ_i at criticality level L_i
J_i	The release jitter of task τ_i
c_j	Core j

Table 1: Real-time System Notation

Assume the platform contains several homogeneous cores, then each core would have identical computing performance. In addition, it is assumed that criticality mode changes do not frequently occur on cores. Based on that, a migratable task is not required to move back and forth continuously between processors.

A sporadic task model is assumed, deadlines are in general constrained; a "job" is used to represent one invocation of a "task".

2.2 Vestal's Algorithm and Run-time Monitoring

Vestal's algorithm [16] allows priorities of high and low criticality tasks to be interleaved in order to provide flexibility in scheduling, however interleaved tasks need to be considered as if they are at the same criticality level. It was later proved that Audsley's priority assignment is still optimal for MCS [10]. The response time analysis for Vestal's algorithm is equation (1) where $hp(i)$ stands for the task set that contains all the tasks which have higher priority than task τ_i.

$$R_i = C_i(L_i) + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(L_i) \tag{1}$$

This and following similar equations are solved using the standard techniques for solving recurrence relations.

But since the original Vestal's algorithm performs pessimistically on criticality inversion cases, Baruah and Burns [3] introduced the usage of a runtime monitor to control the execution time of lower criticality tasks . For example, regarding to task τ_j, which has a higher priority than task τ_i, the execution time C_j shall be:

- if $L_i = L_j$, then $C_j(L_i)$ should be used as the tasks are at the same level of criticality.
- if $L_i < L_j$, then $C_j(L_i)$ should be used since the lower level of assurance is needed for task τ_i.
- if $L_i > L_j$, then task τ_j needs to be guaranteed that it should not execute for more than $C_j(L_j)$ by using a run-time monitor.

The response time analysis is extended from equation (1) to be equation (2):

$$R_i = C_i(L_i) + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(\min(L_i, L_j)) \tag{2}$$

2.3 Adaptive Mixed Criticality Mode

Adaptive Mixed Criticality (AMC) is a further extension by increasing the usage of the run-time monitor [5]. The key thinking in AMC is that lower criticality level tasks may be terminated in order to guarantee higher criticality level tasks complete their executions. Assume the task set has two criticality levels: HI-crit and LO-crit. Then the schedulable test for AMC consists of three phases of analysis. The first phase is to verify the schedulability of LO-crit mode, when all of the tasks are executing with their LO-crit budgets. The response time analysis for this phase is shown in equation (3).

$$R_i(LO) = C_i(LO) + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil C_j(LO) \tag{3}$$

The second phase is to verify the schedulability of HI-crit mode, when only HI-crit tasks are executing and execute with their HI-crit budget. The response time analysis for this phase is shown in equation (4) where $hpH(i)$ stands for the set of HI-crit tasks with higher priority than that of task τ_i.

$$R_i(HI) = C_i(HI) + \sum_{\tau_j \in hpH(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) \tag{4}$$

The third phrase is to check the schedulability of the progress of criticality change. Since exact analysis of this phase is unlikely to be tractable [5], a sufficient analysis can be done by assuming that HI-crit tasks execute in their HI-crit budget while LO-crit tasks execute in their LO-crit budget before the system changes to HI-crit mode. In this case, for HI-crit task τ_i, interferences from HI-crit tasks will not be affected by changing the time when system enters HI-crit mode but interferences from LO-crit tasks will increase if that time increases. So $R_i(LO)$, the time that τ_i finishes all its LO-crit budget, is the latest time point the criticality change may occur. So the response time analysis for this phase is shown in equation (5) where $hpL(i)$ stands for the set of LO-crit tasks with higher priority than that of task τ_i.

$$\begin{aligned} R_i(HI)^* = & C_i(HI) + \sum_{\tau_j \in hpH(i)} \left\lceil \frac{R_i(HI)^*}{T_j} \right\rceil C_j(HI) \\ & + \sum_{\tau_k \in hpL(i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_k(LO) \end{aligned} \tag{5}$$

Note $R_i(HI)^* > R_i(HI)$ so that if the task is deemed schedulable with $R_i(HI)^*$, it is deemed to be schedulable with $R_i(HI)$.

2.4 Semi-partitioned Models

Zaid et al. [1] recently introducing a dual-partitioned scheduling approach, which allows HI-crit tasks to be statically mapped to processors at all times while LO-crit tasks executing with limited migration, under the assumption that both processors will go into HI-crit mode at the same time. Their model consists of two steady modes and a migrating process. In the steady modes (LO-crit mode and HI-crit mode), tasks are fully-partitioned to each core unless a criticality change of the system is detected. During the criticality change, LO-crit tasks can be migrated to other cores to provide flexibility. Thus, a LO-crit task τ_i may have two different designated processors $c_i(LO)$ and $c_i(HI)$. In addition, Zaid et al.'s model also assumed that LO-crit tasks will get a decreased release frequency ($T_i(LO) < T_i(HI)$) but the WCET will remain the same ($C_i(HI) = C_i(LO)$). In all, the response time analysis for two steady modes of each core is shown in equation (6).

$$\begin{aligned} R_i(LO) =& C_i(LO) + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i(LO)}{T_j(LO)} \right\rceil C_j(LO) \\ R_i(HI) =& C_i(HI) + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i(HI)}{T_j(HI)}) \right\rceil C_j(HI) \end{aligned} \tag{6}$$

However, their model only checks the schedualbilities of two states but omits checks on the mode change itself, which makes their results incomplete. In addition, they made an assumption that LO-crit tasks would have increased periods in the HI-criticality mode that decrease the execution rate of these tasks. The scheme presented in this paper has different requirement and analysis.

2.5 Task Allocation

Kelly et al. [12] explored the problem of scheduling MCS tasks on a fixed number of homogeneous processors. Their model focuses on scheduling MCS tasks with relative deadlines equal to periods and tasks are statically allocated to cores. In addition, the bin-packing algorithm they used is First-Fit (FF). FF allocates tasks to the first processor on which it "fits" where "fit" means a task can be successfully scheduled along with the other tasks that are already allocated to that processor. The order of tasks in the task set will affect the performance of First-Fit. It has been demonstrated that task sets with decreasing utilisation order has better performance of FF than other orders in general real-time system. However, as criticality level is included in MCS, decreasing utilisation order is no longer the best choice. Kelly et al. [12] compared the efficiency of FF among different task set orders based on utilisation and criticality. They named two task set orders as Decreasing Utilisation (DU) and Decreasing Criticality (DC). Regarding to DU, tasks with high utilisation values are allocated first. However, because each MCS task is associated with multiple utilisation values, such an ordering requires a single utilisation value to be identified. In their experiment, they used a nominal utilisation $U_i(L_i)$ to present the value of the task's utilisation at the specific criticality level of the task. Regarding to DC, tasks are ordered according to criticality and tasks at the same criticality level are further ordered by decreasing nominal utilisation. After tasks are allocated, they use RM Priority Assignment and Audsley's Optimal Priority Assignment to assign the task priority on each processors for different task set order: RM for DU (DU-RM), Audsley's for DC (DC-Audsley). In their analysis, task sets that are successfully scheduled by either DU-RM or DC-Audsley but not both. So although DC-Audsley outperformed DU-RM according to their experiments, DC-Audsley did not dominate DU-RM.

In addition, First Fit is only one of the algorithms that are used to address the Bin Packing problem. There exists other algorithms which are more suitable than First Fit in certain situations. Among them, Best Fit (BF) and Worst Fit (WF) are two algorithms developed based on FF that not only check all of the previous bins but also bins that are not yet used.

3. SEMI-PARTITIONED MODEL

The aim of the Semi-partitioned method developed in this paper is to schedule more LO-crit task sets than the original Vestal's algorithm and guarantee all of the HI-crit and LO-crit tasks are schedulable if only one core enters its HI-crit mode. In consideration of the need of safety-critical applications, all HI-crit tasks need to be statically allocated to each core. Regarding LO-crit tasks, some may also be statically allocated to cores while others are allowed to migrate to other cores to provide flexibility for mode changes (how to determine whether a task shall be statically allocated or be able to migrate will be addressed in Section IV). Assume that the criticality level change of one core has no effects on others, the basic properties of the model are as follows:

- If all tasks execute within their LO-crit budget then all deadlines are met and no tasks migrate.
- No LO-crit task is allowed to exceed its LO-crit budget.
- If HI-crit tasks on one core exceed their LO-crit budget, then some LO-crit tasks will migrate, but ALL LO-crit tasks and HI-crit tasks remain schedulable.
- If HI-crit tasks on more than one core exceed their LO-crit budget, then some LO-crit tasks will be abandoned, but all HI-crit tasks remain schedulable (without migration).

For example, for a dual-core platform, the dispatching of jobs for execution occurs according to the following rules:

- Each core consists of a criticality level indicator Γ, which is initialised to LO.
- For each core, while ($\Gamma \equiv LO$), task with highest priority is selected for execution.
- If a LO-crit task executes for its LO-crit budget without signalling completion, its current release shall be terminated. If a HI-crit task executes for its LO-crit budget without signalling completion, then the criticality level indicator Γ_i for this core c_i will be changed to HI.
- Once $\Gamma \equiv HI$, if the criticality level indicator of the other core remains in LO, then all HI-crit tasks may execute with their HI-criticality budgets while some LO-crit tasks will keep executing with their LO-crit budgets and other LO-crit tasks will immediately migrate their current release onto the other core. If the

criticality level indicator of the other core is HI, then all of the LO-crit tasks currently executing on the core need to be abandoned while HI-crit tasks execute within their HI-crit budgets.

This only shows the draft idea of the semi-partitioned algorithm, the detailed mechanism of the model will be illustrated in the following sections.

3.1 Dual Core Platform

Assume that a task set S contains several tasks in two criticality levels (HI-crit and LO-crit). If this task set is due to be scheduled on a two cores platform (c_1 and c_2) by the semi-partitioned algorithm, then on each core there shall exist three types of tasks: HI-crit tasks, statically allocated LO-crit tasks and migrating LO-crit tasks. Let HI_i represent the set of HI-crit tasks on core c_i, LO_i to represent the set of statically allocated LO-crit tasks and MIG_i to represent the set of migrating LO-crit tasks, then the following relationship can be obtained:

- $S = (LO_1 \cup LO_2) \cup (HI_1 \cup HI_2) \cup (MIG_1 \cup MIG_2)$

In steady mode, all these tasks are statically partitioned on each core and executing with their LO-crit budgets. Define state X to represent this phrase, then the relationship between tasks and cores may be viewed as:

- $X_1 = LO_1 \cup HI_1 \cup MIG_1$
- $X_2 = LO_2 \cup HI_2 \cup MIG_2$
- $S = X_1 \cup X_2$

If a criticality change occurs on one core (c_i), then HI-crit tasks (HI_i) will execute with their HI-crit budget. For LO-crit tasks, some of them (LO_i) still execute on the core with their LO-crit budget while the others (MIG_i) need to migrate to other cores as there is not enough space for them on the core. Define state $Y(1)$ to represent the case that core c_1 enters its HI-crit mode, then tasks in MIG_1 will be migrated from core c_1 to core c_2 and the relationship between tasks and cores may be viewed as:

- $Y(1)_1 = LO_1 \cup HI_1$
- $Y(1)_2 = LO_2 \cup HI_2 \cup MIG_1 \cup MIG_2$
- $S = Y(1)_1 \cup Y(1)_2$

Define state $Y(2)$ to represent the case that core c_2 enters its HI-crit mode, then tasks in MIG_2 will be migrated from core c_2 to core c_1 and the relationship between tasks and cores may be viewed as:

- $Y(2)_1 = LO_1 \cup HI_1 \cup MIG_1 \cup MIG_2$
- $Y(2)_2 = LO_2 \cup HI_2$
- $S = Y(2)_1 \cup Y(2)_2$

In state $Y(1)$, task set MIG_1 is migrated from core c_1 to core c_2 and in state $Y(2)$, task set MIG_2 is migrated from core c_2 to core c_1. For tasks in $Y(1)_1$ and $Y(2)_2$, the migration progress does not affect their priority orders. For tasks in $Y(1)_2$ and $Y(2)_1$, since extra tasks have been migrated to these two cores, it is likely that the original priority orders will be affected. So the priority orders in $Y(1)_2$ and $Y(2)_1$ may need to be recalculated offline. For these migrated tasks, it is not defined whether they have finished or partly-completed or even not yet started before migration occurs. Assume the migrations have no cost, all of the migrating tasks still need to execute all their LO-crit budget on newly allocated cores in order to guarantee their completion. In addition, these tasks are likely to be released certain time before migrating, which means they have reduced deadlines ($D(r)$) after migration. To compare the exact value of such deadline reduction is unlikely to be tractable as all of the release patterns need to be considered, so a sufficient analysis can be obtained by applying the smallest reduced deadline to each migrating tasks.

Theorem(1): For task τ_i, the worst case after migration is that it needs to execute all its LO-crit budget in a reduced deadline of $D_i^* = D_i - (R_i - C_i)$, where R_i is the worst-case response time for task t_i in state X.

Proof. Assume the latest release of task τ_i is t_0. The task migrates at time $t_0 + t$ and has completed a units of its current release job. Then in order to meet the deadline, task τ_i needs to finish the rest of job, which is $C_i - a$, within time $D_i - ((t_0 + t) - t_0) = D_i - t$ after migration if no migration costs are considered. In addition, for two tasks with same periods, if task τ_1 needs to finish C units in D and task τ_2 needs to finish $C + x$ in $D + x$, then τ_2 is harder to schedule in FPS than τ_1. In other words, if τ_2 is deemed to be schedulable, then τ_1 is also schedulable. Thus, for τ_i, it has worst case when it has to schedule $C_i - a + a = C_i$ within time $D_i - t + a = D_i - (t - a)$. The value of $(t - a)$ may be understood as the time that task τ_i is pre-emptied before migration, and the maximum of this value is $R_i - C_i$ when task τ_i has been pre-emptied for a maximum time without executing any of its job. In all, the worst case for τ_i is that it has to execute all its LO-crit budget (C_i) in a reduced deadline of $D_i^* = D_i - (R_i - C_i)$. □

The above has considered how the semi-partitioned algorithm may improve the scheduling efficiency when only one core has increased to the HI-crit mode. But in reality, both of the cores may be in HI-crit modes at the same time. There are two possible situations in which both of the cores are in HI-crit mode: both cores increases into HI-crit mode at the same time and two cores increases into HI-crit mode one after another. Regarding to the first case, as both cores enters their HI-crit mode, migrating tasks have no place to execute so they need to be abandoned while HI-crit tasks and statically allocated LO-crit tasks are still able to guarantee their completion. Define state BX to represent this case based on state X, the relationship of task sets can be obtained as:

- $BX_1 = LO_1 \cup HI_1$
- $BX_2 = LO_2 \cup HI_2$
- $S = BX_1 \cup BX_2 \cup MIG_1 \cup MIG_2$

Regard to the latter case which is based on the situation that one core has already entered HI-crit mode, that the schedulability of tasks on that core have been covered in the previous section. Referring to the core that enters HI-crit mode later, since extra LO-crit tasks have been executing on the core, only abandoning the migrating tasks may not

guarantee the execution of HI-crit tasks. Considering that, all LO-crit tasks on that core need to be abandoned. Define state $BY(1)$ to represent this situation based on state $Y(1)$ and state $BY(2)$ to represent this situation based on state $Y(2)$, the relationship of task sets can be obtained as:

- $BY(1)_1 = LO_1 \cup HI_1$
- $BY(1)_2 = HI_2$
- $S = BY(1)_1 \cup BY(1)_2 \cup MIG_1 \cup MIG_2 \cup LO_2$
- $BY(2)_1 = HI_1$
- $BY(2)_2 = LO_2 \cup HI_2$
- $S = BY(2)_1 \cup BY(2)_2 \cup MIG_1 \cup MIG_2 \cup LO_1$

3.2 Analysis

The analysis of the semi-partitioned model is quite similar to that of AMC. The schedulable test consists of a three-phase analysis. The first phase is to verify the schedulability of states X_1 and X_2 when all the tasks are partitioned on two cores and executing within their LO-crit budgets. The response time analysis for this phase is shown in equation (7) where $chp(i)$ stands for the set of all tasks with higher priority than that of task τ_i on the same core. So for $X = X_1 \cup X_2$:

$$\forall \tau_i \in X : R_i(LO) = C_i(LO) + \sum_{j \in chp(i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil C_j(LO) \tag{7}$$

The second phrase is to verify the schedulability of states $Y(1)_1$ and $Y(1)_2$ and states $Y(2)_1$ and $Y(2)_2$ in the semi-partitioned model when some tasks have been migrated from one core to another. For states $Y(1)_1$ and $Y(2)_2$, the cores enter HI-crit mode that HI-crit tasks are executing with their HI-crit budgets while LO-crit tasks are executing with their LO-crit budgets (this is represented as $C_i(L_i)$ for task τ_i). The response time analysis for these tasks is shown in equation (8).

$$\forall \tau_i \in (Y(1)_1 \cup Y(2)_2) : \\ R_i(HI) = C_i(L_i) + \sum_{j \in chp(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(L_j) \tag{8}$$

Meanwhile, tasks in $Y(1)_2$ and $Y(2)_1$ are executing in LO-crit mode that all tasks are executing with their LO-crit budgets. Additionally, since migrating tasks are released before moving to the cores, the release jitters of these tasks need to be considered when calculating their interference upon other tasks. In order to guarantee the schedulability, the maximum pre-empty time of the migrating tasks are used as the release jitters when performing response time analysis. According to this, if task τ_i migrates to another core, the release jitter will be $J_i = R_i - C_i(LO)$; otherwise, $J_i = 0$. The response time analysis for these tasks is shown in equation (9).

$$\forall \tau_i \in (Y(1)_2 \cup Y(2)_1) : \\ R_i(LO)^* = C_i(LO) + \sum_{j \in chp(i)} \left\lceil \frac{R_i(LO)^* + J_j}{T_j} \right\rceil C_j(LO) \tag{9}$$

The last phrase is to check the schedulability of the criticality change progress which consists of two parts. The first part is to check the schedulability of cores entering HI-crit mode. This progress is quite similar to the mode change progress from LO-crit level to MID-crit level in three criticality level (LO-crit, MID-crit and HI-crit) AMC [11]. In AMC, LO-crit tasks will be aborted, MID-crit and HI-crit tasks will execute with MID-crit budgets. A sufficient response time analysis for this AMC criticality mode change is represented in equation (10), where $hpM(i)$ stands for the task set that contains all the MID-crit tasks which have higher priority than tasks τ_i and $R_i(LO)$ stands for the response time of the task when the system is in LO-crit mode.

$$\begin{aligned} R_i = & C_i(MID) + \sum_{j \in hpH(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(MID) \\ & + \sum_{k \in hpM(i)} \left\lceil \frac{R_i}{T_k} \right\rceil C_k(MID) \\ & + \sum_{l \in hpL(i)} \left\lceil \frac{R_i(LO)}{T_l} \right\rceil C_l(LO) \end{aligned} \tag{10}$$

In the semi-partitioned model, migration tasks will be "aborted" on the core when migration happens while other LO-crit tasks remains executing in their LO-crit budgets, and HI-crit tasks start to execute with their HI-crit budgets. Comparing with the three criticality level AMC, migrating tasks in semi-partitioned model perform similarly to LO-crit tasks in AMC, other LO-crit tasks in semi-partitioned model performs similarly to MID-crit tasks in AMC but executing with their LO-crit budgets rather than MID-crit budgets, and HI-crit tasks perform similarly to HI-crit tasks but executing with their HI-crit budgets rather than MID-crit budgets. Thus, by modifying equation (10), a sufficient response time analysis for the semi-partitioned model can be obtained as equation (11) where $chpH(i)$ stands for the task set that contains all the HI-crit tasks which have higher priority than task τ_i on the same core, $chpL(i)$ stands for the task set that contains all the non-migrating LO-crit tasks which have higher priority than tasks τ_i on the same core, $chpMIG(i)$ stands for the task set that contains all the migrating LO-crit tasks which have higher priority than tasks τ_i on the same core.

$$\begin{aligned} & \forall \tau_i \in (Y(1)_1 \cup Y(2)_2) : \\ & R_i(HI)^* = C_i(L_i) \\ & \quad + \sum_{j \in chpH(i)} \left\lceil \frac{R_i(HI)^*}{T_j} \right\rceil C_j(HI) \\ & \quad + \sum_{k \in chpL(i)} \left\lceil \frac{R_i(HI)^*}{T_k} \right\rceil C_k(LO) \\ & \quad + \sum_{l \in chpMIG(i)} \left\lceil \frac{R_i(LO)}{T_l} \right\rceil C_l(LO) \end{aligned} \tag{11}$$

The other part of the last phrase is to check the schedulability of migrating tasks. As stated in the semi-partitioned model, these tasks have a reduced deadline for their current release during migrating. As equation (9) represents the response time analysis for these migrating tasks after migration, their results need to be compared with reduced

deadlines to decide their schedulability. Equation (12) shows the calculation of the reduced deadlines for migrating tasks.

$$\forall \tau_i \in (MIG_1 \cup MIG_2) : D_i^* = D_i - (R_i(LO) - C_i) \quad (12)$$

In addition, according to equation (11), it seems that setting LO-crit tasks with high priority is likely to bring more contribution to scheduling other tasks. Furthermore, according to equation (12), tasks with higher priority are likely to have relatively smaller pre-emptive time which leads them to have relatively larger reduced deadline, which makes them easier to schedule. However, such tasks may also have a quite high priority after migration which will bring in more impact on statically allocated tasks, including HI-crit tasks. So there is a payoff when determining the choice of migrating tasks. This is discussed in Section IV.

Referring to the case that both cores increase to their HI-crit mode, the schedulability test for the first situation that both cores enter mode change at the same time have already been coved in equation (8). For the latter case, it is similar to AMC algorithm that all LO-crit tasks needs to be abandoned during mode change. The response time analysis of HI-crit tasks can be viewed as equation (13) :

$$\begin{aligned} R_i(HI)^{**} = & C_i(HI) + \sum_{j \in chpH(i)} \left\lceil \frac{R_i(HI)^{**}}{T_j} \right\rceil C_j(HI) \\ & + \sum_{k \in chpL(i)} \left\lceil \frac{R_i(LO)^*}{T_k} \right\rceil C_k(LO) \end{aligned} \quad (13)$$

This completes all the analysis required to test the schedulability of a dual-criticality system on a dual-core platform. Regarding to all of the equations, the complexity of the semi-partitioned algorithm for a taskset of size n is $O(n^3)$.

3.3 Semi-partitioned Example

To illustrate the response time analysis discussed above, an example will be showed in the following. Regarding the task set in Table 2, τ_1, τ_2, τ_3 and τ_4 are assigned to c_1 and τ_4 may migrate to c_2 when c_1 enters high mode, while τ_5, τ_6, τ_7 and τ_8 are assigned to c_2 and τ_8 may migrate to c_1 when c_2 enters high mode. According to the priorities, for core c_1, $P_3 > P_2 > P_4 > P_1$; for core c_2, $P_7 > P_5 > P_8 > P_6$.

Task	C(LO)	C(HI)	T	D	L	P	c	MIG
τ_1	8	16	36	36	HI	7	1	NO
τ_2	3	4	12	12	HI	3	1	NO
τ_3	1	-	6	6	LO	1	1	NO
τ_4	1	-	12	12	LO	5	1	YES
τ_5	4	5	12	12	HI	4	2	NO
τ_6	10	20	56	56	HI	8	2	NO
τ_7	1	-	9	9	LO	2	2	NO
τ_8	1	-	12	12	LO	6	2	YES

Table 2: AMC Example Task Set

For the semi-partitioned algorithm, the schedulability test will be done in five phases: state X, state $Y(1)$ with the migration progress $X \Rightarrow Y(1)$, state $BY(1)$, state $Y(2)$ with the migration progresses $X \Rightarrow Y(2)$, and state $BY(2)$. For state X, all of the tasks are executing on their LO-crit budget.

- $X_1 = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ and $P_3 > P_2 > P_4 > P_1$
  - $R_3(LO) = 1 < 6 = D_3$
  - $R_2(LO) = 3 + \lceil \frac{4}{6} \rceil \times 1 = 4 < 12 = D_2$
  - $R_4(LO) = 1 + \lceil \frac{5}{6} \rceil \times 1 + \lceil \frac{5}{12} \rceil \times 3 = 5 < 12 = D_4$
  - $R_1(LO) = 8 + \lceil \frac{20}{6} \rceil \times 1 + \lceil \frac{20}{12} \rceil \times 3 + \lceil \frac{20}{12} \rceil \times 1 = 20 < 36 = D_1$
- $X_2 = \{\tau_5, \tau_6, \tau_7, \tau_8\}$ and $P_7 > P_5 > P_8 > P_6$
  - $R_7(LO) = 1 < 9 = D_7$
  - $R_5(LO) = 4 + \lceil \frac{5}{9} \rceil \times 1 = 5 < 12 = D_5$
  - $R_8(LO) = 1 + \lceil \frac{6}{9} \rceil \times 1 + \lceil \frac{6}{12} \rceil \times 4 = 6 < 12 = D_8$
  - $R_6(LO) = 10 + \lceil \frac{23}{9} \rceil \times 1 + \lceil \frac{23}{12} \rceil \times 4 + \lceil \frac{23}{12} \rceil \times 1 = 23 < 56 = D_6$

For state $Y(1)$, task τ_4 has migrated from core c_1 to core c_2, and HI-crit tasks on c_1 will execute on their HI-crit budget while all other tasks remain executing with their LO-crit budget. In addition, task τ_4, as a migrating task, needs to use its reduced deadline for checking its schedulability.

- $Y(1)_1 = \{\tau_1, \tau_2, \tau_3\}$ and $P_3 > P_2 > P_1$
  - $R_3(HI)^* = 1 < 6 = D_3$
  - $R_2(HI)^* = 4 + \lceil \frac{5}{6} \rceil \times 1 = 5 < 12 = D_2$
  - $R_1(HI)^* = 16 + \lceil \frac{36}{12} \rceil \times 4 + \lceil \frac{36}{6} \rceil \times 1 + \lceil \frac{20}{24} \rceil \times 2 = 36 \leq 36 = D_1$ as the worst case happens when $t = R_1(LO) = 20$
- $Y(1)_2 = \tau_4, \tau_5, \tau_6, \tau_7, \tau_8$ and $P_7 > P_5 > P_4 > P_8 > P_6$
  - $D_4^* = 12 - (5 - 1) = 8$
  - $J_4 = 5 - 1 = 4$
  - $R_7(LO)^* = 1 < 9 = D_7$
  - $R_5(LO)^* = 4 + \lceil \frac{5}{9} \rceil \times 1 = 5 < 12 = D_5$
  - $R_4(LO)^* = 1 + \lceil \frac{6}{9} \rceil \times 1 + \lceil \frac{6}{12} \rceil \times 4 = 6 < 8 = D_4(r)$
  - $R_8(LO)^* = 1 + \lceil \frac{7}{9} \rceil \times 1 + \lceil \frac{7}{12} \rceil \times 4 + \lceil \frac{7+4}{12} \rceil \times 1 = 7 < 12 = D_8$
  - $R_6(LO)^* = 10 + \lceil \frac{32}{9} \rceil \times 1 + \lceil \frac{32}{12} \rceil \times 4 + \lceil \frac{32+4}{12} \rceil \times 1 + \lceil \frac{32}{12} \rceil \times 1 = 32 < 56 = D_6$

For state $BY(1)$, core c_2 also increases to HI-crit mode that LO-crit tasks τ_4, τ_7 and τ_8 on this core need to be abandoned while HI-crit tasks τ_5 and τ_6 will execute in their HI-crit budgets. Core c_1 stays unchanged from state $Y(1)$ that no extra checks is required.

- $BY(1)_2 = \tau_5, \tau_6$ and $P_7 > P_5 > P_4 > P_8 > P_6$
  - $R_5(HI)^{**} = 5 + \lceil \frac{5}{9} \rceil \times 1 = 6 < 12 = D_5$ as the worst case happens when $t = R_5(LO)^* = 5$
  - $R_6(HI)^{**} = 20 + \lceil \frac{55}{12} \rceil \times 5 + \lceil \frac{32}{9} \rceil \times 1 + \lceil \frac{32}{12} \rceil \times 1 + \lceil \frac{32}{12} \rceil \times 1 = 55 < 56 = D_6$ as the worst case happens when $t = R_6(LO)^* = 32$

For state $Y(2)$, task τ_8 has migrated from core c_2 to core c_1, and HI-crit tasks on c_2 will execute on their HI-crit budget while all other tasks remain executing with their LO-crit budget. In addition, task τ_8, as a migrating task, needs to use its reduced deadline for checking its schedulability.

- $Y(2)_1 = \{\tau_1, \tau_2, \tau_3\, \tau_4, \tau_8\}$ and $P_3 > P_2 > P_4 > P_8 > P_1$
  - $D_8^* = 12 - (6-1) = 7$
  - $J_8 = 6 - 1 = 5$
  - $R_3(LO)^* = 1 < 6 = D_3$
  - $R_2(LO)^* = 3 + \lceil \frac{4}{6} \rceil \times 1 = 4 < 12 = D_2$
  - $R_4(LO)^* = 1 + \lceil \frac{5}{6} \rceil \times 1 + \lceil \frac{5}{12} \rceil \times 3 = 5 < 12 = d_4$
  - $R_8(LO)^* = 1 + \lceil \frac{6}{6} \rceil \times 1 + \lceil \frac{6}{12} \rceil \times 3 + \lceil \frac{6}{12} \rceil \times 1 = 6 < 7 = D_8(r)$
  - $R_1(LO)^* = 8 + \lceil \frac{23}{6} \rceil \times 1 + \lceil \frac{23}{12} \rceil \times 3 + \lceil \frac{23}{12} \rceil \times 1 + \lceil \frac{23+5}{12} \rceil \times 1 = 23 < 36 = D_1$
- $Y(2)_2 = \{\tau_5, \tau_6, \tau_7\}$ and $P_7 > P_5 > P_6$
  - $R_7(H)^* = 1 < 9 = D_7$
  - $R_5(H)^* = 5 + \lceil \frac{6}{9} \rceil \times 1 = 6 < 12 = D_5$
  - $R_6(H)^* = 20 + \lceil \frac{48}{9} \rceil \times 1 + \lceil \frac{48}{12} \rceil \times 5 + \lceil \frac{23}{12} \rceil \times 1 = 48 \leq 56 = D_6$ as the worst case happens when $t = R_6(LO) = 23$

For state $BY(2)$, core c_1 also increases to HI-crit mode that LO-crit tasks τ_4, τ_7 and τ_8 on this core need to be abandoned while HI-crit tasks τ_5 and τ_6 will execute in their HI-crit budgets. Core c_2 stays unchanged from state $Y(2)$ that no extra checks is required.

- $BY(2)_1 = \tau_1, \tau_2$ and $P_3 > P_2 > P_4 > P_8 > P_1$
  - $R_2(HI)^{**} = 4 + \lceil \frac{4}{6} \rceil \times 1 = 5 < 12 = D_5$ as the worst case happens when $t = R_2(LO)^* = 4$
  - $R_1(HI)^{**} = 16 + \lceil \frac{36}{12} \rceil \times 4 + \lceil \frac{23}{6} \rceil \times 1 + \lceil \frac{23}{12} \rceil \times 1 + \lceil \frac{23}{12} \rceil \times 1 = 36 \leq 36 = D_1$ as the worst case happens when $t = R_1(LO)^* = 23$

As all of the sufficient response time analysis above are less or equal to their deadline, this task set is schedulable by using the semi-partitioned algorithm.

If the non-migration algorithm is applied to schedule this task set, neither core c_1 nor c_2 is schedulable since $R_1 = 16 + \lceil \frac{44}{12} \rceil \times 4 + \lceil \frac{44}{6} \rfloor \times 1 + \lceil \frac{44}{12} \rfloor \times 1 = 44 > 36 = D_1$ and $R_6 = 20 + \lceil \frac{57}{12} \rceil \times 5 + \lceil \frac{57}{9} \rceil \times 1 + \lceil \frac{57}{12} \rceil \times 1 = 57 > 56 = D_6$. As migration only occurs if necessary, and the above example shows multi-core platform delivers improved schedulability if follows that the semi-partitioned algorithm dominates any non-migration algorithm.

3.4 Returning to LO-crit Mode

This section will address the issue of returning to LO-crit mode. There are three possible cases based on the number of cores in HI-crit mode and which core is returning to LO-crit mode.

- The first case occurs when only one core is in its HI-crit mode. In this case, once the core (core c_1 in state $Y(1)$ and core c_2 in state $Y(2)$) in HI-crit mode experiences an idle tick, it can return to LO-crit mode and the next release of migrating tasks will be on their original processor.
- The second case may happen when both cores are in HI-crit mode and the core which enters HI-crit mode later is returning to LO-crit mode. In this case, once the core (core c_2 in state $BY(1)$ and core c_1 in state $BY(2)$) in HI-crit mode experiences an idle tick, it can return to LO-crit mode and all of the tasks previously abandoned, including the allocated LO-crit tasks and migrating tasks belong to both cores, will start to execute on this processor.
- The last case happens when both of the cores are in HI-crit mode and the core which enters HI-crit mode earlier (including the case that both core enters HI-crit mode at the same time) is returning to LO-crit mode. In this case, once the core (core c_1 in state $BY(1)$, core c_2 in state $BY(2)$, and either core c_1 or c_2 in state BX) in HI-crit mode experiences an idle tick, all of the migrating tasks belongs to both cores will start executing on this processor.

4. SEMI-PARTITIONED CONFIGURATION

The previous section has described how to determine whether a given set of tasks with fixed priorities and allocated cores is schedulable by the semi-partitioned algorithm. This section will consider how to apply the semi-partitioned algorithm to allocate a set of tasks to a dual-core platform. Given that bin-backing algorithms will be used for task allocation, the first step is to sort the set of the tasks into a specified order. Since the primary target of MCS is to guarantee the execution of HI-crit tasks, it will be efficient to check if all the HI-crit tasks are schedulable first, which means it is helpful to put all of the HI-crit tasks in front of LO-crit tasks. As stated in Section 2.5, criticality-aware utilisation descending order provides better performance than others in First-Fit partitioned MCS. It can be assumed that criticality-aware utilisation descending order may also provide good performance in semi-partitioned MCS as the majority of tasks are still partitioned. However, different task orders perform differently under different bin-packing algorithms. So other possible task sorting orders, such as criticality-aware period descending order and criticality-aware deadline descending order, are required to be evaluated.

According to the migration mechanism stated in the semi-partitioned model, setting a task as migratable will add extra computation load to the system as these tasks need to continue on the destination core, so it will be better to minimise the chance of setting a task as migratable. Considering that, the next step is to check whether the task set is schedulable with the non-migration algorithm. The semi-partitioned algorithm will only be applied when the non-migration algorithm cannot schedule the task set. The non-migration algorithm simply assigns tasks using First-Fit bin packing algorithm and checks response times when HI-crit tasks execute with HI-crit budgets and LO-crit tasks with LO-crit budgets. Note that only LO-crit tasks are migratable in the semi-partitioned algorithm, so if the non-migration algorithm is not able to schedule all of the HI-crit tasks then the task set will not be schedulable by the semi-partitioned algorithm.

Regarding to the semi-partitioned algorithm, there are several possible approaches based on the choice of bin packing algorithm and selection of migration tasks. As stated in Section 2.5, First-Fit, Best-Fit and Worst-Fit are the three

mostly used bin-packing algorithms in MCS. It is not clear which method will perform best in for the semi-partitioned algorithm. For the choice of migration tasks, there are two main options. One option is to simply set the current fetched task migratable while the other option attempts to migrate the "highest" priority LO-crit task. The "highest" priority task here stands for the task with the highest priority among all migratable tasks. So this option attempts to set LO-crit tasks migratable starting with the highest priority.

For priority assignment, Audesly's optimal priority scheme will be used to assign priorities for each task. An important issue is that migration tasks will be assigned two priorities: one used in its original core, the other for its destination core. These values will be determined during task assignment. A detailed semi-partitioned approach is as follows:

1. A task is fetched from the sorted task set.
2. Assign the task to one of the cores according to the chosen bin packing algorithm (FF or BF or WF).
3. Uses Audsley's algorithm to assign priorities for all tasks and check whether all of the tasks are schedulable.
4. If an un-schedulable task is found, try to assign the task to the other core and assign the priority order and do the checking again.
5. If both cores have been checked and neither of them can schedule the fetched task, setting the fetched task/tasks migratable will be considered.
6. Assign the fetched task to one of the cores according to the chosen bin packing algorithm (FF or BF or WF) and set the task migratable based on one of the approaches.
7. Uses Audsley's algorithm to assign priorities for all tasks on both cores considering migration effects and check whether all of the tasks on both cores are schedulable.
8. If an un-schedulable task is found, try to assign the task to the other core and do the checking again.
9. If both cores have been checked and still neither of them can schedule the fetched task, then the task is not schedulable by the semi-partitioned algorithm.
10. Fetch another task to start a new loop until the task set is empty or a task is detected unschedulable.

5. EVALUATION

According to the model, the semi-partitioned algorithm should have better performance than the non-migration algorithm based on the original Vestal's algorithm. But it is uncertain by how much the semi-partitioned algorithm has improved the scheduling efficiency. This section will first introduce an experiment designed to compare the performance of the semi-partitioned algorithm and the non-migration algorithm. Then it will explore six semi-partitioned approaches and attempt to find the best one.

In order to represent the performance of the semi-partitioned algorithm, software is produced to check the performance of different semi-partitioned approaches and the non-migration algorithm with the AMC algorithm. The software consists of three parts. The first part of the software will generate task sets. Tasks are randomly set to be HI-crit tasks or LO-crit tasks but the percentage of HI-crit tasks is controlled to be a fixed number, and for all HI-crit tasks, their HI-crit WCETs are a fixed number of times of their LO-crit WCETs. These two fixed values will be changed in the experiments to explore the performance of different algorithms among different task set settings. In order to gain uniform distributed parameters, UUnifast-discard algorithm [8] is used to generate "nominal" utilisation (a "nominal" utilisation represents the LO-crit utilisation for a LO-crit task or the HI-crit utilisation for a HI-crit task), and Log-uniform algorithm [14] is used to generate periods. Other parameters of each task can be calculated based on these two values ($D = T, C(L_i) = U_i(L_i) * T$). The second part of the software is to pre-sort each task set before scheduling. As stated in the task allocation section, all tasks will be sorted in criticality-aware utilisation descending order. In such order, HI-crit tasks will be placed in front of all LO-crit tasks, and both HI-crit tasks and LO-crit tasks are in utilisation descending order independently. The last part of the software is to test the success rate of scheduling the task sets by the different scheduling algorithms.

We investigated the performance of six semi-partitioned approaches (Table 3) and compared them with the non-migration algorithm. The non-migration algorithm is chosen as the lowest bound of performance. Figure 1 shows the percentage of tasksets that are schedulable for a system of 12 tasks, with on average 50% of those tasks having HI-crit (P=0.5) and HI-crit execution budget twice that of LO-crit (f=2). The Y-axis shows the percentage of the successful task sets while the X-axis shows the sum of nominal utilisations of each task set. The sum of utilisation ranges from 1.7 to 2.1 to amplify the results.

Notation	Description
Non-migration	The non-migration approach
Semi1FF	Semi-partitioned approach that migrates the fetched task and uses First Fit bin packing algorithm
Semi1BF	Semi-partitioned approach that migrates the fetched task and uses Best Fit bin packing algorithm
Semi1WF	Semi-partitioned approach that migrates the fetched task and uses Worst Fit bin packing algorithm
Semi2FF	Semi-partitioned approach that migrates the "highest" priority task and uses First Fit bin packing algorithm
Semi2BF	Semi-partitioned approach that migrates the "highest" priority tasks and uses Best Fit bin packing algorithm
Semi2WF	Semi-partitioned approach that migrates the "highest" priority tasks and uses Worst Fit bin packing algorithm

Table 3: Real-time System Notation

We observe that all of the semi-partitioned schedulability tests outperform the non-migration algorithm by a considerable margin. This is expected as the semi-partitioned algorithms allows more LO-crit tasks to be scheduled. Comparing all of the semi-partitioned algorithms, the algorithms that migrate allocated tasks (Semi2) perform slightly bet-

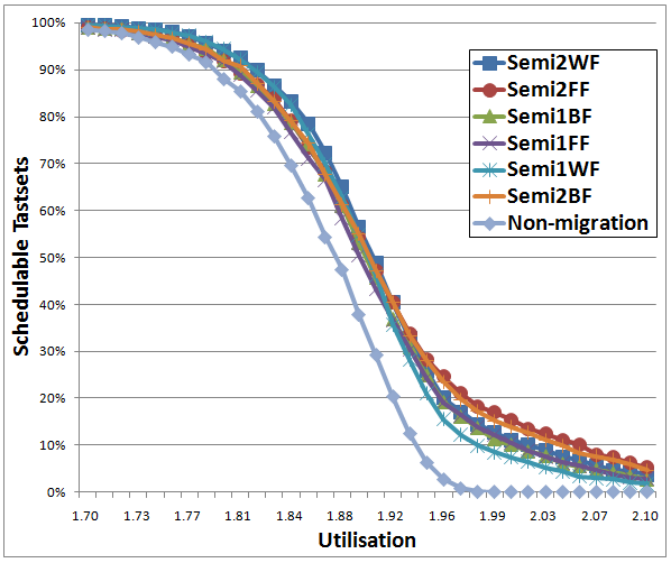

Figure 1: Percentage of Schedulable Tasksets

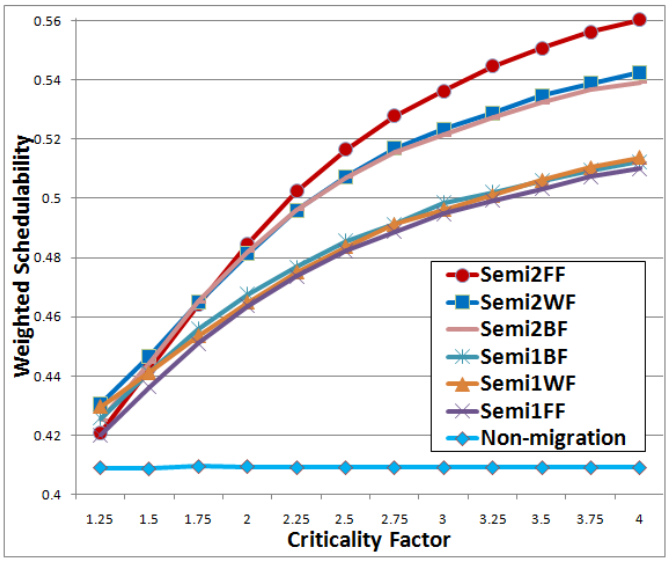

Figure 2: Varying the Criticality Factor

ter than those algorithms which only migrate fetched tasks (Semi1) . This is also to be expected as the former type of algorithm checks more possibilities and is more likely to migrate LO-crit tasks with higher priorities, which, as discussed earlier, may improve scheduling. Within Semi2 algorithms, Semi2WF has the best performance when the sum of utilisation is smaller than 1.9 while Semi2FF outperforms others in the other cases.

In order to explore the performances of the algorithms relating to criticality factor $(C(HI)/C(LO))$ and the percentage of HI-crit tasks. Weighted schedulability measure $W_y(p)$ [6] is used for schedulability test y as a function of parameter p to reduce a 3-dimensional plot to 2 dimensions. For each value of p, this measure combines results for all tasksets τ generated for all of a set of equally spaced utilisation levels (1.6 to 2.2 in steps of 0.012).

$$W_y(p) = (\sum_{\forall \tau} u(\tau) * S_y(\tau, p))/\sum_{\forall \tau} u(\tau) \qquad (14)$$

We show how the results are changed by varying one key parameters at a time. Figure 2 varies the criticality factor, Figure 3 varies the percentage of HI-crit tasks and Figure 4 varies the size of the taskset. The x-axis stands for the parameter examined and y-axis represents the weighted value. According to Figure 2, Semi2WF has the best performance when criticality factor is smaller than 2 while Semi2FF outperforms others in the rest of the cases. In addition, all semi-partitioned algorithms have increased performance as the criticality factor increases. This is to be expected as the increase of WCET difference between different criticality levels allows more scheduling potential for migration tasks. According to Figure 3, the performance of the semi-partitioned algorithms have formed a U-shape curve since each end of the interval represents a one-criticality task set, and hence the priorities are optimal. Regarding to individual performances, Semi2WF has the best performance in most of the cases $(0 < p < 0.8)$ while Semi1WF has the best performance when the percentage of criticality tasks is quite high $(0.8 < u < 1)$. According to Figure 4, the performance of the semi-partitioned algorithms have formed a U-shape curve. This is expected as tasks are relatively large in small sized tasksets which adds difficulty in finding acceptable migrating tasks, while in large sized tasksets, the interference from high priority tasks increases which adds difficulty to the schedulability of migrated tasks with reduced deadlines. Regarding to individual performances, Semi2WF has the best performance when the size of the taskset is small $(n \leqslant 12)$ while Semi2FF has the best performance in the rest of the cases $(n > 12)$.

Overall, Semi2WF and Semi2FF has the best performance in the majority of the cases. Thus, a combining usage of Semi2WF and Semi2FF may be the most appropriate on scheduling a two criticality level MCS on a dual-core platform.

6. CONCLUSION

In this paper, we considered semi-partitioned scheduling, upon a preemptive dual-core platform, with two criticality levels and with the assumption that if only one core enters its HI-crit mode, all tasks remained schedulable. Moreover we require that if both cores enter the HI-crit mode then all HI-crit tasks are guaranteed to meet their deadlines. Finally we note that with normal behaviour (i.e. both cores are in the LO-crit mode) no migrations occur.

We have studied six scheduling approaches based on two possible task migration scheme and three bin-packing algorithms. All of these algorithms have better performance than the non-migration algorithm, and the semi-partitioned approaches that migrates the highest priority migratable LO-crit tasks with Worst Fit bin-packing algorithm and the semi-partitioned approaches that migrates the highest priority migratable LO-crit tasks with First Fit bin-packing algorithm have been observed to have better performance than others in most of the cases. It is suggested that a combined use of these two approaches may deliver the most advantage of the semi-partitioned approach for a mixed criticality system with two levels of criticality executing on a dual-core platform.

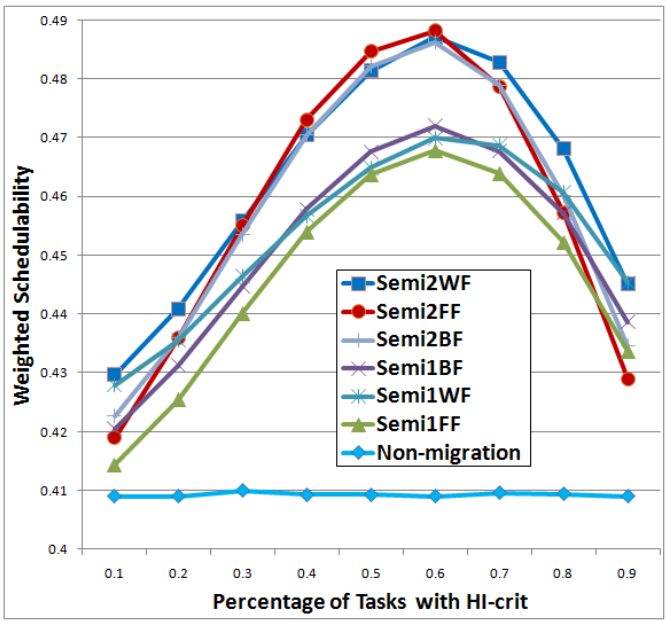

Figure 3: Varying the Criticality Percentage

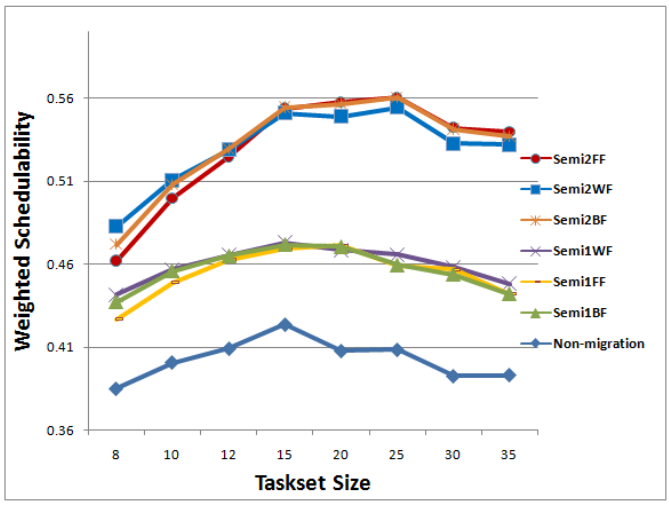

Figure 4: Varying the Taskset Size

In future work, we will generalise the analysis to multiple criticality levels; we shall also extend the model to platforms contains more than two cores.

7. REFERENCES

[1] Z. Al-bayati, Q. Zhao, A. Youssef, H. Zeng, and Z. Gu. Enhanced partitioned scheduling of mixed-criticality systems on multicore platforms. In *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, pages 630–635. IEEE, 2015.

[2] J. H. Anderson, S. K. Baruah, and B. Brandenburg. Multicore operating-system support for mixed criticality. In *Proceedings of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification*. Citeseer, 2009.

[3] S. Baruah and A. Burns. Implementing mixed criticality systems in ada. In *Reliable Software Technologies-Ada-Europe*, pages 174–188. Springer, 2011.

[4] S. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 13–22. IEEE, 2010.

[5] S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Real-Time Systems Symposium*, pages 34–43. IEEE, 2011.

[6] Andrea Bastoni, Björn Brandenburg, and James Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. *Proceedings of OSPERT*, pages 33–44, 2010.

[7] A. Burns. System mode changes-general and criticality-based. In *WMC*, pages 3–8. RTSS, 2014.

[8] A. Burns and R. I. Davis. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 47(1):1–40, 2010.

[9] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys*, 43(4):35, 2011.

[10] F. Dorin, P. Richard, M. Richard, and J. Goossens. Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Real-time systems*, 46(3):305–331, 2010.

[11] T. Fleming and A. Burns. Extending mixed criticality scheduling. *Proc. WMC, RTSS*, pages 7–12, 2013.

[12] O. R. Kelly, H. Aydin, and B. Zhao. On partitioned scheduling of fixed-priority mixed-criticality task sets. In *Trust, Security and Privacy in Computing and Communications*, pages 1051–1059. IEEE, 2011.

[13] R. Love. Kernel korner: CPU affinity. *Linux Journal*, 2003(111):8, 2003.

[14] R. Staggord P. Emberson and R. I. Davis. Techniques for the synthesis of multiprocessor tasksets. In *1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time System (WATERS)*.

[15] P. Rodriguez, L. George, Y. Abdeddaim, and J. Goossens. Multi-criteria evaluation of partitioned edf-vd for mixed-criticality systems upon identical processors. In *Workshop on Mixed Criticality Systems*, 2013.

[16] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 239–243. IEEE, 2007.