Utilising Asymmetric Parallelism in Multi-Core MCS Implemented via Cyclic Executives

Tom Fleming
Department of Computer Science,
University of York, UK.
Email: tdf506@york.ac.uk

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

***Abstract*—Near future real-time systems are faced with the inevitability of multi-core architectures. Theoretically, multi-core architectures bring an increase in platform resources and speed, however, this comes at the cost of the predictability fundamental to real-time systems. With the increase in platform speed and resources comes the added pressure to incorporate more and more functionality into a common hardware platform. In this work we consider a Mixed Criticality (MC) approach applicable to all future, but particularly near future systems, which must support sequential high criticality code alongside less critical (perhaps parallelised) applications. We propose a simple design objective to reduce the number of cores high criticality work executes on, with the aim to increase the predictability of these applications by requiring verification over a fewer number of cores. We investigate how an increase in predictability might provide a reduction in Worst Case Execution Time and use Integer Linear Programming (ILP) to reduce the number of cores utilised by high criticality tasks as far as possible.**

I. Introduction

It is becoming more and more evident that future and near future real-time systems will be required to execute on powerful multi-core hardware platforms. This requirement is born initially of necessity, as single-core architectures are being abandoned for multi-core, however, many are seeing the opportunity to utilise these powerful platforms to combine previously federated functionality. By integrating functionality onto the same hardware platform, inevitably, the situation arises where applications of differing levels of criticality must execute alongside each other. This poses the challenge of providing the suitable level of separation to the higher criticality work, such that it may be guaranteed that no lower criticality work may effect its execution. Providing such mechanisms is fundamental for the certification and validation of Mixed Criticality systems.

Complicating the matter further is the simple usage of multi-core platforms themselves. Typical real-time applications of a high level of criticality are, and have been for a long time, residents of the single-core/processor domain. The introduction of multiple cores brings problems such as interference between tasks, controlling memory access and inter-core communications to name a few. In addition to these issues, when considering a mixed criticality system, one must also ensure the sufficient isolation of criticality levels, such that non of the above factors may have impact from a lower to a higher criticality level. For example, a lower criticality application may not cause contention on the shared bus and adversely effect the execution of a higher criticality task such that its execution cannot be guaranteed within safe bounds. However, multi-core platforms promise improvement over previous single-core architectures. In addition to increased processing capacity, they offer improved power usage and thermal output due to lower individual clock speeds. Finally, they provide a platform amenable to parallelised applications which may benefit lower criticality work, particularly that involved in tasks such as run-time simulation and image processing.

With all these advantages and drawbacks in mind, we propose a design optimisation to, in some way, ease the transition from single-core to multi-core architectures. Our work is based on the well known Cyclic Executive paradigm [1] where a barrier protocol [5] is used to strictly separate the execution of differing levels of criticality. We utilise this separate notion of criticality level execution to pursue the goal of reducing the number of cores the highest criticality (HI) work may execute on. The aim being that with a reduction in the number of cores, simpler verification with fewer overheads will be available. We illustrate this optimisation and its impact with an example and provide experimental work investigating the reduction in the number of cores for high criticality work using synthetic task sets. This work aims to present the reasoning and advantages of reducing the number of cores for higher criticality level execution.

The remainder of this paper is structured as follows: Section II described the system model and reviews prior work on ILP, Section III presents and discusses the notion of limiting the number of cores for HI criticality execution, Section IV provides some insight via an example, Section V provides an evaluation and Section VI poses some concluding remarks.

II. The System Model

Throughout this work, we make reference to features of general systems and our mixed criticality cyclic executive model. This section will define and clarify the notation used.

While this work is applicable to mixed criticality applications with greater than two levels of criticality, we restrain ourselves to just two. We make use of the Mixed Criticality

model proposed by Vestal [7] with a minor alteration. In the standard model a task has a WCET value assigned for its own criticality level and all those below, we simply consider each task to only ever have two WCET values, one at its own criticality level and one at the lowest criticality level in the system. In general, we define a standard mixed criticality (dual criticality) task set as $\tau_i = \{C_i(LO), C_i(HI), T_i, D_i, L_i\}$, where $C_i(LO)$ is the Worst Case Execution Time (WCET) of a task in the LO criticality mode, $C_i(HI)$ is the WCET of a task in the HI criticality mode, T_i is the period, D_i is the deadline and L_i is the criticality level.

When discussing the Mixed Criticality Cyclic Executive we define T^M as the major cycle length. This major cycle is comprised of a number of minor cycles T^F. The minor cycles execute in order during each major cycle while the major cycle repeats cyclically upon completion of the previous cycle. Figure 1 shows an example set-up of 4 minor cycles within a major cycle.

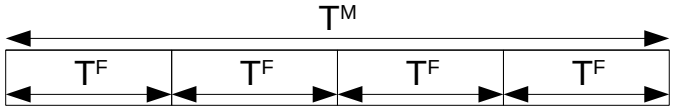

Fig. 1. The Cyclic Executive Structure.

In addition to the somewhat standard CE practice described above, we utilise a barrier mechanism [5] to account for mixed criticality workloads. The barrier completely separates the execution of criticality levels across all cores during a given minor cycle. Thus at runtime: HI criticality work is executed first (up to their $C(LO)$ values), once all HI work has completed on all CPU cores the barrier is triggered, this releases LO criticality work for execution. A criticality change occurs when all HI work, executing to their $C(LO)$ predictions, has not completed by the predicted LO invocation of the barrier. In this situation, HI criticality work may execute up until its $C(HI)$. Be this as it may, in reality, the barrier protocol occurs when HI work has completed, regardless of whether it is before or after a criticality change. It provides a level of dynamic behaviour such that we do not discuss the *dropping* of low criticality tasks, but due to the statically ordered nature of the cyclic executive, in the worst case, they would simply not execute. Finally, if a criticality change does occur, the criticality level of the system may be reduced at the start of the next major cycle.

Integer Linear Programming is used in this work to solve our allocation problem. We will briefly review the construction of a standard ILP model as presented in [4], extensions to allow for task splitting are presented in [3] but are omitted from this work. We will describe how the model is constructed based on the .lp (intended to be more user readable) format provided by the Gurobi [6] LP solver[1].

Fundamental to our ILP models is the representation of possible task locations as binary variables, these are defined in the following format:

- $T[taskNumber]\_[core][minorCycle]$

For a dual criticality system the model is structured in the following way:

–Maximise/Minimise: The first section contains the objective function, in the case of our standard ILP model this is left blank as we only seek to establish feasibility.

–Constraints: Arguably the largest and most significant section, the constraints in our ILP models are split broadly in two.

- *Group One:* The first group ensures that tasks are allocated correctly based on their periods. In a system with four minor cycles per major cycle, a task, τ_i, where $T_i = T^F$, would require the following constraints:

  $T_i\_11 + T_i\_21 = 1$
  $T_i\_12 + T_i\_22 = 1$
  $T_i\_13 + T_i\_23 = 1$
  $T_i\_14 + T_i\_24 = 1$

  However, if $T_i = T^M$ the constraint would be:

  $T_i\_11 + T_i\_21 + T_i\_12 + T_i\_22 + T_i\_13 + T_i\_23 + T_i\_14 + T_i\_24 = 1$

- *Group Two:* The constraints in group two seek to ensure that any allocation is also schedulable. The WCETs are used in conjunction with the location variables to check feasibility. If a task is allocated to a location, the variable will be set to 1 and thus the relevant WCET is included. In a dual criticality system, this is achieved in three stages: *Stage One:* Ensure the schedulability of the HI criticality tasks using their $C(HI)$ values. For HI criticality tasks τ_i and τ_l:

  $C_i(HI) \times T_i\_11 + C_l(HI) \times T_l\_11 \leq T^F$
  $C_i(HI) \times T_i\_21 + C_l(HI) \times T_l\_21 \leq T^F$

  Stage Two: Ensure the schedulability of the HI tasks executing to their $C(LO)$ estimates. In addition, an X variable in included to represent the spare capacity available for LO criticality tasks to execute. The same X variable is used across all cores during any given minor cycle, providing a means of modelling the barrier protocol. For the same HI criticality tasks τ_i and τ_l:

  $C_i(LO) \times T_i\_11 + C_l(LO) \times T_l\_11 + X_1 \leq T^F$
  $C_i(LO) \times T_i\_21 + C_l(LO) \times T_l\_21 + X_1 \leq T^F$

  Stage Three: Ensure the scheduability of the LO criticality tasks. This must ensure that their $C(LO)$ values fit within the relevant X variable for each minor cycle. Given two LO criticality tasks τ_z and τ_x:

  $C_z(LO) \times T_z\_11 + C_x(LO) \times T_x\_11 - X_1 \leq 0$
  $C_z(LO) \times T_z\_21 + C_x(LO) \times T_x\_21 - X_1 \leq 0$

–Bounds: This section simply states the upper and lower bounds placed on our variables. As all but the X variables are integers, only they are bounded to be $0 \leq X \leq T^F$.

[1]For a more detailed overview see [4] & [3]

–Variable Types: Finally, the last section is where variable types are declared, all but the X variables are declared as binaries.

III. Reducing the number of cores used in HI criticality execution

The general scheduling of real-time task sets on multi-core platforms is a challenging topic. Many COTS (Consumer Off The Shelf) hardware platforms contain ambiguous and often poorly document features which do not facilitate real-time software. Often these hardware platforms are designed with the *average* case in mind, rather than the *worst case*. Knowledge of execution and performance is fundamental to real-time system design, as without verification and certification, such software is not fit for purpose. An important part of this problem, is that verification becomes more difficult as the number of cores in use is increased. Often this leads to an increased level of pessimism in WCET predictions as increasingly large margins are added for safety, due to the unpredictable nature of the platform.

When Mixed Criticality is added to the issue, a balance is required between pessimism for safety, but effective use of the, new, larger, multi-core platform. While higher criticality work seeks to maintain the same guarantees of timeliness as it did in the single-core era, new lower criticality applications are able to make use of the multi-core architecture. Coupled with these conflicting requirements is the necessity to support legacy code, designed for single-core sequential execution. The development of software is expensive, companies often utilise legacy and even commercial code to fulfil the needs of the system. As such platforms treat these applications in a *black box* fashion, in this situation re-design for a more parallel architecture isn't possible.

In order to, at least partially, deal with this issue we propose a simple design goal. Given a multi-core computing platform, the number of cores that any higher criticality work is executing upon should be reduced to the minimum possible. We consider this goal in relation to our Mixed Criticality Cyclic Executive model where work of differing criticality levels is temporally separated, thus removing the chance for lower criticality levels to interfere in some way with the execution of higher levels. We envisage a scenario where HI criticality work executes first, on a very small number of cores (perhaps only one or two), the remaining LO criticality work is then allowed to execute using the complete system resources (4,8,16 cores). Such a situation allows both the safety critical sequential (legacy) HI criticality code to execute alongside LO criticality, perhaps highly parallelised code designed for new multi-core platforms.

In short, the advantages of such an approach are twofold. Firstly, with a reduction in the number of cores used for HI criticality work, we increase the predictability in its execution. The increase is due to, in essence, reducing the number of factors that must be considered during analysis, and thus increasing the ease in which the performance of a system may be understood. Secondly, as a result of the first point, we are able to provide a reduction in the WCET estimates for HI criticality tasks. The WCETs may be reduced as a direct result of the increase in predictability.

While it is difficult to reason about exactly how much more difficult analysis becomes as additional cores are included at the HI criticality level, we may make an assumption in order to illustrate how reducing the number of cores used by the HI criticality work is beneficial to the system as a whole. To this end, one might assume that, perhaps optimistically, for every core which does not execute any HI criticality tasks, all HI criticality tasks have their WCETs reduced by 10%. This serves the purpose of simulating the increase in predictability and accuracy of the analysis and therefore the reduction in pessimism required for verifiably safe WCETs.

To satisfy the requirements described above our model must implement two key features.

1) We must provide an optimisation to reduce the number of cores HI criticality work is executing on as far as possible.
2) We model the improved analysability of a task by reducing its WCET by 10% for each core not executing HI criticality work.

To achieve these goals we first define a variable for each core in the system to represent whether HI criticality work is allocated. The variable P_z, where z is the core, is a binary variable which is set equal to 1 if no HI criticality work is allocated and 0 if HI criticality work is allocated. As such our optimisation goal is simply to maximise P, and thus reduce the number of cores HI criticality work is executing on as much as possible.

To model the 10% reduction in WCET for each core unused by HI criticality work we must modify the binary variables used to represent HI criticality tasks. These variables become Integers, for example $0 \leq T_i\_11 \leq 10$, they must be less than or equal to 10 and greater than or equal to 0. Setting a variable to 0 still indicates that the task is not scheduled in that location, however a positive value now indicates that a task is allocated to that location. A value of 10 indicates that all cores in the system are used to execute HI criticality work, whereas a value of 9 indicates one core does not execute any HI criticality tasks.

As a scheduled task has a value from 1 to 10, when including the task in the WCET constraints we multiply the variable by the WCET / 10. As such if all cores are used for HI criticality tasks, and the variable is equal to 10, then $10 \times WCET/10$ will include the full WCET with no reduction. However if one core is unused by HI criticality work, the variable will equal 9, thus $9 \times WCET/10$ models a 10% reduction in WCET as a result. The WCET constraint for T_i_11 was:

$$C_i(HI) \times T_i\_11$$

It now becomes:

$$\frac{C_i(HI)}{10} \times T_i\_11$$

An additional constraint ensures that for each core unused by HI criticality tasks, the location variable is reduced by 1. Consider the HI criticality task τ_i and two processor variables P_1 & P_2.

$$T_i\_11 + T_i\_21 + P_1 + P_2 = 10$$

The task is scheduled in one of the two variables (T_i_11&T_i_21). As the constraint must equal 10, the variable will be reduced, from 10, by the number of processors with no HI criticality execution. Thus if $P_1 = 1$ (no HI criticality execution on core 1) and $P_2 = 0$ (HI criticality execution is present on core 2) then the variable containing the task location must equal 9. As such, 90% of the task is included in the WCET constraints, modelling a 10% reduction.

The optimisation is driven by a set of constraints which only allow P_z to be set to 1 if no HI criticality work is allocated to that core. An example of this is shown for core 1.

$$T_i\_11 + T_i\_12 + T_i\_13 + T_i\_14+$$
$$T_l\_11 + T_l\_12 + T_l\_13 + T_l\_14 + 80 \times P_1 \leq 80$$
...

Here the HI criticality variables which represent possible locations for tasks τ_i & τ_l as summed with $80 \times P_z$. If any location contains a HI criticality task, P_z may not be set equal to 1, thus indicating that HI criticality work is present on that core. The value 80, used to both multiply P_z and constrain the inequality is calculated by multiplying the number of location variables in the constraint $\times 10$. Thus ensuring that even if tasks are located in every variable, the constraint is still satisfied, but P_z may only equal 1 if no HI criticality work is allocated on that core.

We will briefly go over the changes to each section of the constraints.

- Periodicity Constraints

The changes to these constraints are perhaps the most significant in structure. As the original variables were binaries, adjusting the constraints for our integer location variables required a re-work. We make use of a familiar trick, where a task + a binary variable $\times 10$ is constraint to be less than or equal to 10. We introduce a number of B variables to facilitate this. Given two task locations:

T_i_11

T_i_21

We define two constraints:

$$T_i\_11 + 10 \times B_1 \leq 10$$
$$T_i\_21 + 10 \times B_2 \leq 10$$

We require this task to be scheduled in one of the two variables, thus we constrain the B variables:

$$B_1 + B_2 = 1$$

The sum of the B variables is required to equal 1, thus one task variable must contain no execution to allow for the B variable to equal 1. The actual value the task variable takes is controlled in a later constraint, but if it is allocated to that location the variable will be greater than or equal to 1. The complete set of constraints for a task where $T_i = T^F$ as as follows:

$$T_i\_11 + 10 \times B_1 \leq 10$$
$$T_i\_21 + 10 \times B_2 \leq 10$$
$$T_i\_12 + 10 \times B_3 \leq 10$$
$$T_i\_22 + 10 \times B_4 \leq 10$$
$$T_i\_13 + 10 \times B_5 \leq 10$$
$$T_i\_23 + 10 \times B_6 \leq 10$$
$$T_i\_14 + 10 \times B_7 \leq 10$$
$$T_i\_24 + 10 \times B_8 \leq 10$$
$$B_1 + B_2 = 1$$
$$B_3 + B_4 = 1$$
$$B_5 + B_6 = 1$$
$$B_7 + B_8 = 1$$

- WCET Constraints

The constraints which use the WCET values of each task to ensure scheduability must simply be altered to include the WCET / 10 for each HI criticality task. This is required for both stage one (HI WCET):

$$\frac{C_i(HI)}{10} \times T_i\_11 + \frac{C_l(HI)}{10} \times T_l\_11 \leq T^F$$
$$\frac{C_i(HI)}{10} \times T_i\_21 + \frac{C_l(HI)}{10} \times T_l\_21 \leq T^F$$

And for stage two (LO WCETs of HI criticality tasks):

$$\frac{C_i(LO)}{10} \times T_i\_11 + \frac{C_l(LO)}{10} \times T_l\_11 + X_1 \leq T^F$$
$$\frac{C_i(LO)}{10} \times T_i\_21 + \frac{C_l(LO)}{10} \times T_l\_21 + X_1 \leq T^F$$

- Additional constraints

Additional constraints are added firstly to ensure the P_z variables correctly represent which cores are executing HI criticality tasks. The complete set of constraints are shown for τ_i and τ_l on two cores:

$$T_i\_11 + T_i\_12 + T_i\_13 + T_i\_14+$$
$$T_l\_11 + T_l\_12 + T_l\_13 + T_l\_14 + 80 \times P_1 \leq 80$$
$$T_i\_21 + T_i\_22 + T_i\_23 + T_i\_24+$$
$$T_l\_21 + T_l\_22 + T_l\_23 + T_l\_24 + 80 \times P_2 \leq 80$$

Finally constraints are included to reduce the value in the location variable for each core HI criticality work is not allocated to. These constraints are show for τ_i where $T_i = T^M$:

$$T_i\_11 + T_i\_21 + T_i\_12 + T_i\_22+$$
$$T_i\_13 + T_i\_23 + T_i\_14 + T_i\_24 + P_1 + P_2 = 10$$

And where $T_i = T^F$:

$$T_i\_11 + T_i\_21 + P_1 + P_2 = 10$$
$$T_i\_12 + T_i\_22 + P_1 + P_2 = 10$$
$$T_i\_13 + T_i\_23 + P_1 + P_2 = 10$$
$$T_i\_14 + T_i\_24 + P_1 + P_2 = 10$$

Finally it should be noted that, while these constraints and variables have changed for HI criticality tasks, the variables and constraints of LO criticality tasks have not changed. LO criticality tasks still make use of binary variables to indicate their location.

IV. Example

In this section we will describe the process in more detail using our Mixed Criticality Cyclic Executive model. We will make use of an example set of tasks, and illustrate through example schedules how the design optimisation might work. The ILP model from [4] will be used as a base to allow for the optimisation.

The task set shown in Table I will be used as our example. This task set contains 10 tasks, 5 of HI criticality and 5 of LO.

τ	$C(LO)$	$C(HI)$	T	L_i
τ_1	5	10	25	HI
τ_2	5	10	25	HI
τ_3	5	10	25	HI
τ_4	10	15	50	HI
τ_5	15	20	100	HI
τ_6	5	-	25	LO
τ_7	5	-	25	LO
τ_8	5	-	25	LO
τ_9	10	-	50	LO
τ_{10}	10	-	100	LO

TABLE I
An example set of tasks.

Initially, we seek any acceptable schedule of the task set on a 3 core platform with 4 minor cycles. Each minor cycle has a length of 25 ($T^F = 25$) and the major cycle has a length of 100 ($T^M = 100$). An ILP model is produced to solve this problem and ascertain the initial feasible solution. The resulting non-optimised schedule is shown in Figure 2.

With the dashed vertical line within each minor cycle representing the last possible time at which the barrier switches from the HI to LO work without triggering a criticality change, it is clear that the HI criticality work is spread across all cores during each minor cycle. In addition, it is also apparent that there is a significant amount of free capacity in LO mode (to the right of the barrier). By optimising the schedule the aim would be to use this spare capacity to reduce the number of cores HI criticality work is scheduled upon.

Figure 3 shows the results of such an optimisation. This solution restricts HI criticality work to cores 2 and 3 while after the barrier protocol, LO tasks are able to utilise all cores, particularly during minor cycles 2 and 4 where the additional core is required to schedule τ_9 . Thus our optimisation was able to reduce the number of cores used by HI criticality work from 3 to 2.

V. Evaluation

This evaluation aims to provide a broad picture as to the performance of our scheme using ILP. The setup for the evaluation is as follows:

- Each task set contained 20 tasks.
- The target platform has 4 cores.
- Task utilisation were generated using UUniFast [2] providing an unbiased distribution of utilisation values.
- Task periods were randomly selected from 25, 50 and 100 (as $T^F = 25$ & $T^M = 100$).
- Deadlines were set equal to periods. $D_i = T_i$.
- The execution time of a task must account for criticality, as such: $C_i(L_i) = U_i/T_i$
- For all but the execution time of the task at its own criticality level, execution times were determined by $C_i(L_v) = C_i(L_i) * CF$ CF is the criticality factor, a random variable between X and Y.
- Criticality levels within each task set were evenly distributed.
- A reduction in WCET of 10% per core HI criticality work does not execute on is included to simulate a decrease in pessimism when fewer cores are used.

The results shown in Figure 4 are a standard plot of schedulability (eg. Number or % of scheduleable task sets) as the utilisation of the generated task sets is increased. The x axis continues to 400% utilisation as the system contains 4 cores. This plot illustrates that the ability to reduce the pessimism in the WCET predictions does provide a small improvement to schedulability.

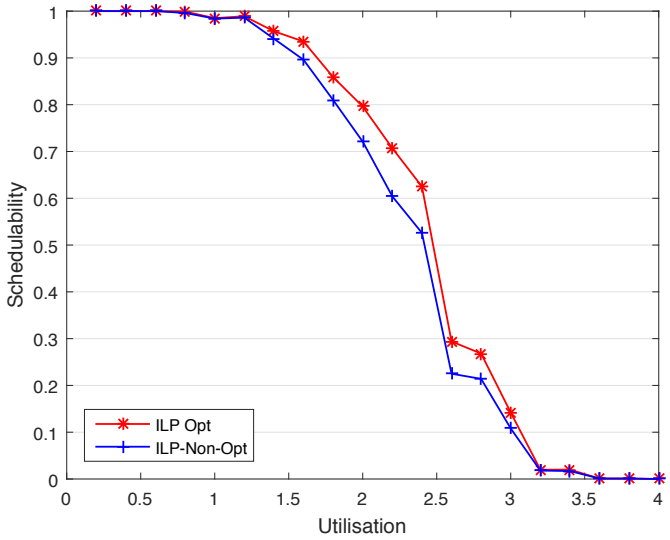

Fig. 4. A standard schedulability plot illustrating the difference between the optimised and non-optimised ILP models..

Figure 5 illustrates the average number of cores used by HI criticality work (Y axis) as the utilisation is increased. The results show that the non-optimised ILP implementation very quickly uses all 4 cores to schedule its HI criticality work, whereas the optimised ILP model increases more gradually as the utilisation of the task sets is increased.

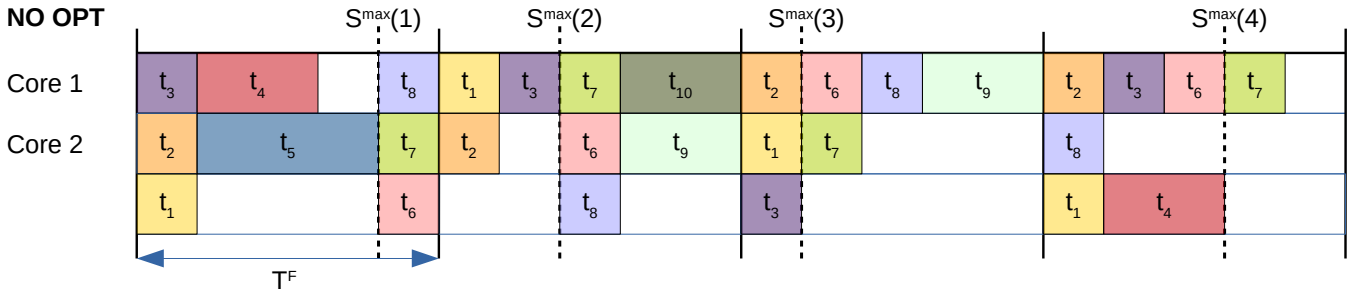

Fig. 2. A standard (non-optimised) schedule of the task set in Table I.

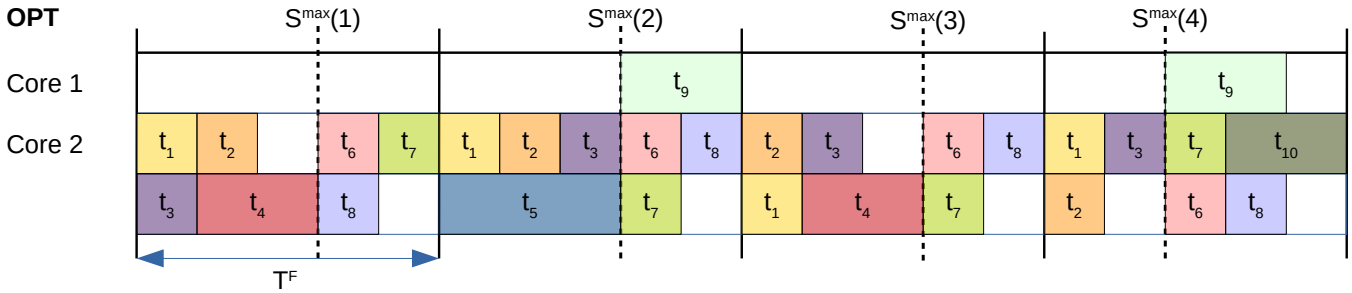

Fig. 3. An allocation, optimised to minimise the number of cores for HI criticality tasks.

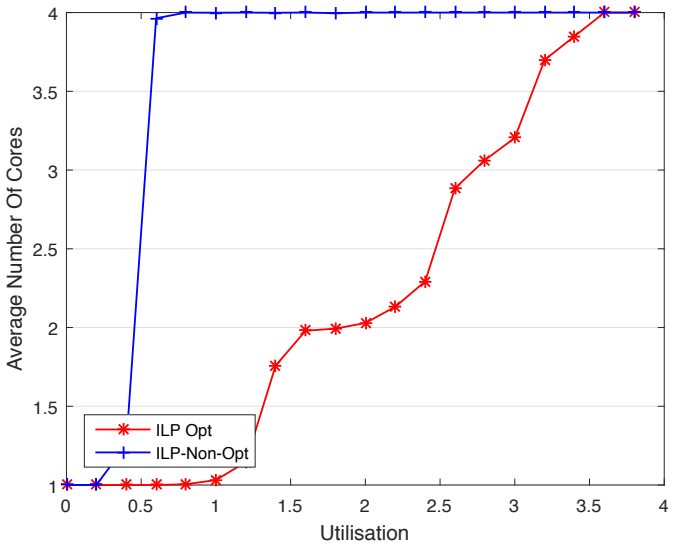

Fig. 5. A plot showing the average number of cores used by HI criticality tasks at each given utilisation.

VI. Conclusion

In conclusion, in this work we have put forward the notion of reducing the number of cores work at the HI criticality level execute upon. We reason about the desire to support the conflicting requirements of sequential (legacy) and parallised code, and argue that by reducing the number of cores, the level of pessimism and difficulty in analysing HI criticality tasks can be reduced. We propose a model which provides a 10% reduction in HI criticality task's WCET for each core not used by HI criticality tasks. This was illustrated through the use of an example set of tasks, showing how in our cyclic executive system an allocation may be optimised to this end. Finally, we used experimental data to illustrate how reducing the WCET, in line with a reduced number of cores used for HI criticality execution, provides an increase in overall schedulability.

Acknowledgements

The authors acknowledges the support and funding provided for this work by BAE Systems, and the ESPRC (UK) via MCC & MCCps grants.

References

[1] T. Baker and A. Shaw. The cyclic executive model and ada. In *Real-Time Systems Symposium, 1988., Proceedings.*, pages 120–129, Dec 1988.

[2] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[3] T. Fleming, S. Baruah, and A. Burns. Improving the schedulability of mixed criticality cyclic executives via limited task splitting. In *Proceedings of the 24rd International Conference on Real Time and Networks Systems*, RTNS '16, 2016.

[4] T. Fleming and A. Burns. Investigating mixed criticality cyclic executive schedule generation. In *Proceedings of Workshop on Mixed Criticality, IEEE Real-Time Systems Symposium (RTSS)*, 2015.

[5] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. Scheduling of mixed-criticality applications on resource-sharing multicore systems. In *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, pages 1–15, Sept 2013.

[6] I. Gurobi Optimization. Gurobi optimizer 6.0. http://www.gurobi.com/.

[7] 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, dec. 2007.