Investigating Mixed Criticality Cyclic Executive Schedule Generation

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*—Mixed Criticality systems require a difficult compromise to be drawn between efficient system utilisation and sufficient separation of critical components. In addition to these challenges, hardware platforms are becoming increasingly multi-core in nature bringing up additional scheduling issues. Previous publications have met these challenges by suggesting a Cyclic Executive based approach for Mixed Criticality scheduling. They make use of a barrier protocol to separate the execution within each minor cycle, executing higher critical work, then less critical work. The barrier protocol allowed such a separation of criticalities to remain consistent across all cores in a given platform. This strict separation has the advantage that higher criticality work cannot suffer interference from lower, including communication and recourse access. One of the key challenges of using a Cyclic Executive is the construction of a valid schedule. In this work we consider the question, "Is it worth using an optimal solver such as Integer Linear Programming (ILP) for Cyclic Executive schedule generation?". We start by extending the Cyclic Executive model to include multiple minor cycles. An ILP model is described and evaluated against the heuristic worst fit. The results show that ILP significantly outperforms worst fit. Finally we show that ILP is not only effective, but also efficient in terms of runtime and scalability for the examples and parameters considered in this work, making it a practical choice for Cyclic Executive schedule generation of real systems.**

I. Introduction

With the introduction of powerful multi-core architectures comes the desire to consolidate functionality, that was previously spread across many nodes, onto a single common hardware platform. Inevitably such a consolidation gives rise to the situation where more critical work must be placed upon the same resources as less critical work. This consolidation has brought about the notion of a Mixed Criticality (MC) system. Scheduling such a system is challenging as highly critical work must often be certified and follow safety standards such as Design Assurance Levels (DAL) in the aerospace industry and ASIL (Automotive Safety Integrity Levels) in the automotive industry. Care must be taken to ensure that less critical work can not interfere with the execution of higher critical work. The design of such systems becomes a trade-off between efficiently utilising the system resources while providing an adequate level of separation to satisfy any safety requirements.

One of the most widely used scheduling policies in industry is the Cyclic Executive (CE). Such policies execute code cyclically in a pre-defined order, as such they are highly deterministic which makes them a favourable choice for highly critical applications with stringent certification requirements. Cyclic Executive systems are made up of a major cycle which is composed on a number of minor cycles, the major cycle repeats in a cyclic manner. Naturally this determinism comes with some drawbacks, Baker and Shaw [1] performed some initial evaluations on the CE model, they noted some restrictions:

- Cyclic Executives can only easily support periodic work.
- Tasks must have periods that are multiples of the minor cycle.
- Tasks must have deadlines equal to or less than the minor cycle.
- Tasks cannot have a period greater than the major cycle.

In addition to these drawbacks, the creation of CE schedules is well known to be NP-hard. Despite these issues the high level of determinism makes Cyclic Executives popular schedulers.

Baruah and Burns [3] investigate the notion of a mixed criticality cyclic executive. In order to provide the separation required between different levels of criticality they use the scheme proposed by Ginnopoulou et al. [6]. This approach uses a barrier protocol to completely separate the execution of different criticality levels. The barrier mechanism works by having each CPU call it when its execution for a particular criticality level has completed. Once all CPUs have called the barrier, they are released and allowed to execute the work for the next criticality level. The barrier protocol requires minimal hardware or OS support. Baruah and Burns make use of this protocol within a CE context. Within each minor cycle, work is executed in order of criticality, highest criticality first, each level is separated by a barrier.

Burns et al. [5] build upon the work in [3] by considering the creation of CE schedules using heuristics to allocate tasks to cores. This work considers the simple case where a system is made up of a single minor cycle (i.e. minor cycle = major cycle), they assess the performance of First Fit (FF), Worst Fit (WF) and First Fit with Branch and Bound (FFBB). They show that the barrier protocol does impact the ability to create CE schedules but conclude this it is a necessary compromise to allow for more robust systems that are easier to certify.

In this work we seek to extend the investigation into cyclic executive schedule construction. We will extend our view of the system model to include multiple minor cycles within a major cycle. Alongside heuristic based techniques there are optimal solutions which typically come with increased execution overheads. As we know that the MC scheduling problem is NP-hard in the strong sense [2] we extend our work to consider an optimal solver. One such technique is Integer Linear Programming (ILP), in this work we make use of ILP and show that not only does it allow a large number of task sets to be scheduled, even in the extended system model but for the purposes of CE schedule constriction it is efficient.

Throughout this work we will make use of the MC system model proposed by Vestal in 2007 [8]. Vestals model proposes that each task in a system has a WCET value for its own criticality level and all those below. Our model is made up of a number of dual criticality periodic tasks with the properties $\tau = \{C(LO), C(HI), T, D, L\}$ where $C(LO)$ is the LO criticality WCET, $C(HI)$ is the HI criticality WCET (and $C(HI) > C(LO)$), T is the period, D is the deadline and L is the criticality level (HI or LO). We use T^F to denote the minor cycle and T^M to denote the major cycle.

In addition to the model above, we assume a constrained system where the major cycle is a multiple of the minor cycle. As such task periods must also be multiples of the minor cycle and no greater than the major cycle. In general in this work we consider 4 minor cycles per major cycle, where $T^F = 25$ and $T^M = 100$. We do not consider the issue of assigning an arbitrary set of periods to a cyclic executive.

The remainder of this work is structured as follows, Section II will detail the construction of an ILP model used to check for feasible CE schedules and show its effectiveness against heuristic based techniques, Section III will consider whether ILP can be practically used by assessing its computational overheads and Section IV will present some conclusions to this work.

II. Using ILP to create CE schedules

In this section we will describe how the constraints of a cyclic executive can be expressed as an ILP model and how this model is used to check for a feasible schedule. We extend the work of [3] by allowing for multiple minor cycles within the major cycle, this leads to an allocation problem of tasks to frames and the cores within each frame. We will briefly recap the runtime of a mixed criticality CE using the barrier protocol and its schedulability test, following this the ILP implementation will be described by means of an example.

The runtime of a single minor cycle in a (dual criticality) mixed criticality cyclic executive is as follows:

CE Runtime:

- The minor cycle begins by executing HI criticality tasks on all cores.
- Once tasks on a core have finished executing HI work they signal the barrier protocol.
- If all cores signal the barrier before their $C(LO)$ execution times, LO criticality work may commence.
- If any core does not signal completion by their $C(LO)$ then the system moves into the HI criticality mode and all HI tasks are allowed to execute up to their $C(HI)$ execution times.

The latest time at which each core could call the barrier protocol to report HI work complete is denoted by $S(i,j)$, where i is the core and j is the minor cycle. The point at which the system changes from executing the HI work in the LO mode to executing the LO work is denoted by $S^{max}(j)$, where j is the minor cycle. As such the schedulability of a mixed criticality CE can be determined as follows (where $HI(i,j)$ is the set of high criticality tasks scheduled on core i, minor cycle j and $LO(i,j)$ is the set of LO criticality tasks scheduled on core i, minor cycle j):

1) HI criticality tasks must fit within the minor cycle:
$$\forall i \text{ and } j, \sum_{k \in HI(i,j)} C_k(HI) \ \leq T^F.$$
2) The value of $S(i,j)$ can be calculated for each core:
$$S(i,j) \ = \sum_{k \in HI(i,j)} C_k(LO)$$
3) The value of $S^{max}(j)$ used across all CPUs is:
$$S^{max}(j) = \max(S(i,j))$$
4) LO criticality jobs must fit within the time between $S^{max}(i,j)$ and the end of the minor cycle:
$$\forall i \text{ and } j, \sum_{k \in LO(i,j)} C_k(LO) \leq T^F - S^{max}(j)$$

We will describe the construction of our ILP model via the use of an example. Consider the task set shown in Table I:

τ	$C_i(LO)$	$C_i(HI)$	T_i	D_i	L_i
$T1$	3	4	25	25	HI
$T2$	4	5	50	50	HI
$T3$	5	6	50	50	HI
$T4$	13	15	25	25	HI
$T5$	10	-	25	25	LO
$T6$	2	-	50	50	LO
$T7$	3	-	25	25	LO
$T8$	5	-	100	100	LO

TABLE I
A Mixed Criticality task set with 4 minor cycles ($T^F = 25, T^M = 100$).

The construction of the ILP model to check the schedulability of the task set in Table I will now be described based on a 2 core platform. We will describe this model based on the syntax of the Gurobi optimiser [7] which is the tool used throughout this work.

In order to achieve our goal of testing for any valid CE schedule we require a simplistic method of modelling our system. In order to model the possible locations of a task we create a variable for each location, the variables are in the format $T[tasknumber]\_[core][cycle]$. Each variable is declared as a binary value, therefore if it is set to 1 the task is scheduled in that location. For example $T1$ has a period of 25, as such is included in all 4 of the minor cycles, therefore it must be scheduled on one of the two cores within each minor cycle, (T_11 or T_21 where $T^F = 1$). In the next part of the model we define the bounds for these variables so that each task might only be scheduled the correct number of times in the correct places (no duplicate tasks etc.).

With this in mind we construct our maximize statement, the first section of the ILP model. As we are not interested in optimising any particular parameter, we need not include anything in this section. We merely seek to discover if a scheduleable assignment exists, we require at least one feasible schedule.

The second stage of the model is the *Subject To* section, in this section the key constraints of the model are defined. To model this we set-up a constraint for each minor cycle, for cycle one $T1$ would have the constraint $T1\_11 + T1\_21 = 1$, this ensures that $T1$ is only scheduled on one of the two cores in the first minor cycle. These constraints are repeated for the remaining minor cycles.

If a task has a period of 50, it must be scheduled once in the first two minor cycles and once in the second two. We use the same notation to model this, for example cycles one and two for $T2$ can be constrained with the following: $T2\_11 + T2\_21 + T2\_12 + T2\_22 = 1$.

Finally if a task has a period of 100 then it must be scheduled only once within the major cycle, we model this for $T8$ as follows: T8_11 + T8_21 + T8_12 + T8_22 + T8_13 + T8_23 + T8_14 + T8_24 = 1.

The complete set of constraints for the task set shown in Table I are shown below:

```
Subject To
T1_11 + T1_21 = 1
T1_12 + T1_22 = 1
T1_13 + T1_23 = 1
T1_14 + T1_24 = 1
T2_11 + T2_21 + T2_12 + T2_22 = 1
T2_13 + T2_23 + T2_14 + T2_24 = 1
T3_11 + T3_21 + T3_12 + T3_22 = 1
T3_13 + T3_23 + T3_14 + T3_24 = 1
T4_11 + T4_21 = 1
T4_12 + T4_22 = 1
T4_13 + T4_23 = 1
T4_14 + T4_24 = 1
T5_11 + T5_21 = 1
T5_12 + T5_22 = 1
T5_13 + T5_23 = 1
T5_14 + T5_24 = 1
T6_11 + T6_21 + T6_12 + T6_22 = 1
T6_13 + T6_23 + T6_14 + T6_24 = 1
T7_11 + T7_21 = 1
T7_12 + T7_22 = 1
T7_13 + T7_23 = 1
T7_14 + T7_24 = 1
T8_11 + T8_21 + T8_12 + T8_22 +
T8_13 + T8_23 + T8_14 + T8_24 = 1
```

Statements are now required to ensure that the taskset is schedulable in the configuration chosen. This is done in three stages.

Stage One: The first stage aims to ensure that the HI criticality work is schedulable when it executes up to its maximum $C(HI)$ WCET value. This is done by multiplying each tasks WCET with the variables representing the possible locations of the tasks. If the variable is set to 1 and the task is scheduled, then the answer will be equal to the WCET, if 0 then the answer is 0. The notation for the HI criticality tasks in the HI mode is shown below:

```
4 T1_11 + 5 T2_11 + 6 T3_11 + 15 T4_11 <= 25
4 T1_21 + 5 T2_21 + 6 T3_21 + 15 T4_21 <= 25
4 T1_12 + 5 T2_12 + 6 T3_12 + 15 T4_12 <= 25
4 T1_22 + 5 T2_22 + 6 T3_22 + 15 T4_22 <= 25
4 T1_13 + 5 T2_13 + 6 T3_13 + 15 T4_13 <= 25
4 T1_23 + 5 T2_23 + 6 T3_23 + 15 T4_23 <= 25
4 T1_14 + 5 T2_14 + 6 T3_14 + 15 T4_14 <= 25
4 T1_24 + 5 T2_24 + 6 T3_24 + 15 T4_24 <= 25
```

Stage Two: The second stage checks the schedulability of the HI criticality tasks executing to their LO WCET values, this is done in the same way as stage one. In addition to this an X value is added to the calculation, one X value per minor cycle (X_1, X_2, X_3, X_4). The X value represents the time between the point at which all cores complete their execution of the HI criticality tasks (S^{max}), and the end of the minor cycle (T^F).

```
3 T1_11 + 4 T2_11 + 5 T3_11 + 13 T4_11 +X_1<= 25
3 T1_21 + 4 T2_21 + 5 T3_21 + 13 T4_21 +X_1<= 25
3 T1_12 + 4 T2_12 + 5 T3_12 + 13 T4_12 +X_2<= 25
3 T1_22 + 4 T2_22 + 5 T3_22 + 13 T4_22 +X_2<= 25
3 T1_13 + 4 T2_13 + 5 T3_13 + 13 T4_13 +X_3<= 25
3 T1_23 + 4 T2_23 + 5 T3_23 + 13 T4_23 +X_3<= 25
3 T1_14 + 4 T2_14 + 5 T3_14 + 13 T4_14 +X_4<= 25
3 T1_24 + 4 T2_24 + 5 T3_24 + 13 T4_24 +X_4<= 25
```

Stage Three: The final stage seeks to ensure than the LO criticality tasks are schedulable within the time X we calculated above. This is achieved by a similar process to stages one and two, but this time also subtracting X. The solution must be less than or equal to 0 for the LO criticality execution to be schedulable within X.

```
10 T5_11 + 2 T6_11 + 3 T7_11 + 5 T8_11 −X_1<= 0
10 T5_21 + 2 T6_21 + 3 T7_21 + 5 T8_21 −X_1<= 0
10 T5_12 + 2 T6_12 + 3 T7_12 + 5 T8_12 −X_2<= 0
10 T5_22 + 2 T6_22 + 3 T7_22 + 5 T8_22 −X_2<= 0
10 T5_13 + 2 T6_13 + 3 T7_13 + 5 T8_13 −X_3<= 0
10 T5_23 + 2 T6_23 + 3 T7_23 + 5 T8_23 −X_3<= 0
10 T5_14 + 2 T6_14 + 3 T7_14 + 5 T8_14 −X_4<= 0
10 T5_24 + 2 T6_24 + 3 T7_24 + 5 T8_24 −X_4<= 0
```

The model then declares any bounds required, as all but 4 of the variables used are declared as binaries only 4 bounds are defined. The X values are bounded to be less than or equal to 25, in reality these variables should never reach this point.

```
Bounds
X_1 <= 25
X_2 <= 25
X_3 <= 25
X_4 <= 25
```

Finally we declare all variables used.

```
Binaries
T1_11  T1_21  T1_12  T1_22  T1_13  T1_23  T1_14  T1_24
T2_11  T2_21  T2_12  T2_22  T2_13  T2_23  T2_14  T2_24
T3_11  T3_21  T3_12  T3_22  T3_13  T3_23  T3_14  T3_24
T4_11  T4_21  T4_12  T4_22  T4_13  T4_23  T4_14  T4_24
T5_11  T5_21  T5_12  T5_22  T5_13  T5_23  T5_14  T5_24
T6_11  T6_21  T6_12  T6_22  T6_13  T6_23  T6_14  T6_24
T7_11  T7_21  T7_12  T7_22  T7_13  T7_23  T7_14  T7_24
T8_11  T8_21  T8_12  T8_22  T8_13  T8_23  T8_14  T8_24

Integers
X_1  X_2  X_3  X_4

End
```

In order to access the performance of the ILP model we compared it against the heuristic Worst Fit (WF) which performed well in the experimentation undertaken in [5]. In the work of [5] WF performed its allocation in two stages:

- **Stage One** Allocate the HI criticality tasks and locate point S^{max}.
- **Stage Two** Allocate the LO criticality tasks in the time remaining, $T^F - S^{max}$.

As the prior work dealt with the simpler single cycle model, the implementation of worst fit used in this work had to take into account the multi-cycle system. This is done as follows:

- **Stage One** Allocate the HI criticality tasks to minor cycles.
- **Stage Two** Allocate the LO criticality tasks to minor cycles.
- **Repeat For Each Minor Cycle**
  Stage One Allocate HI criticality tasks assigned to the minor cycle to cores and locate point S^{max}.
  Stage Two Allocate the LO criticality tasks assigned to the minor cycle in the time remaining, $T^F - S^{max}$.

Worst fit and ILP were compared by means of experimental data using randomly generated task sets. The parameters for this experimentation were as follows:

- Our experiments were based on a 4 core platform.
- Each task set consisted of 20 tasks.
- 10,000 task sets were generated at each 5% utilisation interval.
- Tasks were generated as follows: utilisations (U) were uniformly generated via UUniFast [4], periods were selected from the set $\{25, 50, 100\}$, $C(LO)$ values were created by, $C(LO) = U \times T$, $C(HI)$ values were created by multiplying $C(LO)$ values via a random value between 1.1 and 1.9.
- The criticality levels within a task set were evenly distributed.
- CE execution is split across 4 minor cycles where $T^F = 25$ and $T^M = 100$.
- Tasks may have periods of 25, 50 and 100. These are allocated randomly during taskset generation.

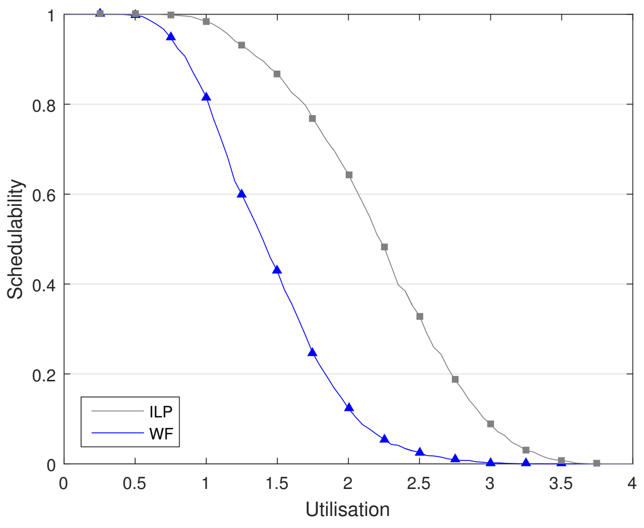

Fig. 1. The effectiveness of WF and ILP to generate CE schedules.

Figure 1 is typical of the results found over a range of parameters and shows that ILP significantly out performs the WF heuristic. Where ILP still boasts schedulability at nearly 0.8 WF is only able to manage around 0.1. ILP provides a clear improvement over the heuristic based techniques. With ILP being an optimal solver this improvement is somewhat expected, the question remains, are the overheads of using ILP in comparison to WF worth the increase in the number of CE schedules generated.

On an alternate note, if ILP is used in the constrained CE model from [5] where $T^F = T^M$ then the results are interesting.

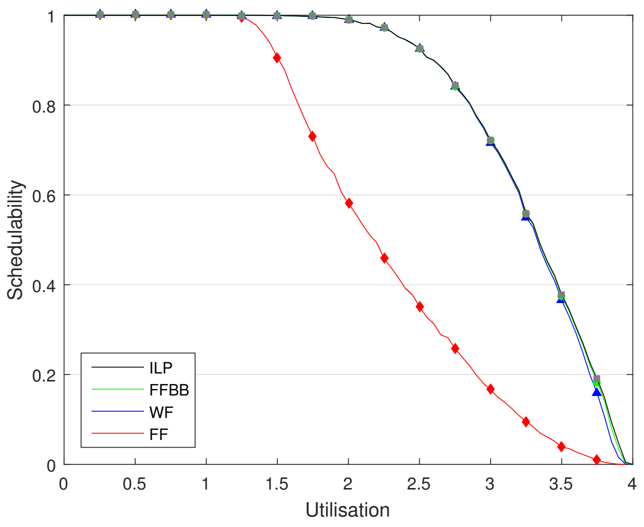

Fig. 2. A comparison between ILP and prior heuristics presented in [5].

Figure 2 shows that for the restricted case of a single minor cycle, the heuristics FFBB[1] and WF perform extremely close to the ILP solution. This makes it clear that these heuristics are very well optimised for this problem, but the additional complexity of multiple minor cycles is a significant issue. This is likely down to it becoming a two stage allocation process for the heuristic techniques (allocate to minor cycles then to cores within those cycles).

III. The efficiency of using ILP for CE schedule creation

As established in Section II, ILP can provide significant improvements for CE schedule creation, especially when a more complex CE model is considered. However ILP based solutions are well known for having high computational overheads. This section will investigate the overheads involved with our technique and show how ILP can be effectively used for schedule generation.

In order to investigate the computation time required for ILP in comparison to WF we made use of the inbuilt timing tools in Matlab. This provides a comparable baseline which allows the real world performance of each approach to be assessed. Our test platform consisted of a 32 core (AMD Opteron 6134) compute server. We timed the execution of the complex test with the aim of investigating the cost of the additional schedulability provided by ILP.

The figures in Table II show the average time taken to execute (in seconds), per task set for both ILP and WF. The parameters of this experiment were: 20 tasks per set with evenly distributed (dual) criticality levels.

	WF	ILP
Average Time (sec)	0.010	0.0125

TABLE II
The average execution time of WF and ILP.

While these results do show that on average the ILP solver takes longer than WF to solve the CE schedule generation problem, both times are negligible when you consider real world use. During experimentation thousands of task sets are tested in order to produce results, while the experiments take some time to run, a single set of tasks can be checked quickly. In a real use case it is likely that only a single version of the system need be checked at any given time and both WF and ILP take, on average, a very small amount of time to solve a single problem. ILP however boasts greatly increased schedulability, we observed an average increase of 0.19 and up to 0.53 at some utilisations. The question remains, if this is true for a system with 20 tasks, how do both approaches fair when the number of tasks are increased? In other words, is it scalable?

[1]Inital allocation is performed by First Fit, the largest and smallest S^{max} values are identified, these are used to perform a Branch and Bound search to attempt to minimise S^{max}. See [5] for details.

This was explored by re-running the experiments and varying the number of tasks from 20 to 100, the results of this are shown in Figure 3:

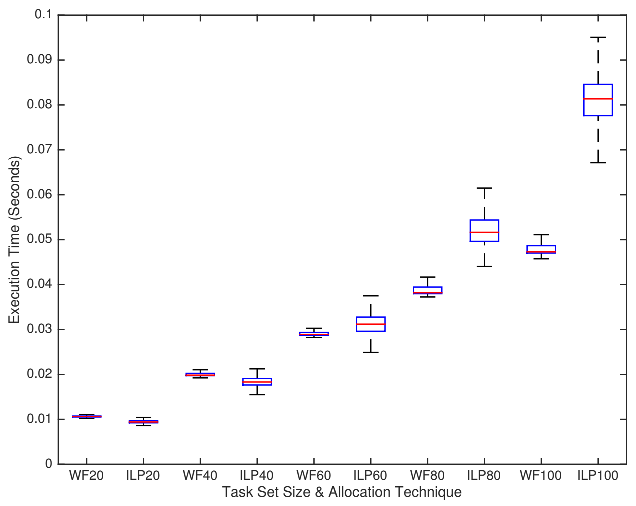

Fig. 3. The increase in computation time as the number of tasks per set is increased.

The results in Figure 3 show data for more than 99.99% of the task sets tested, the majority of the outliers not shown here completed within 4 seconds[2]. Although a rise in the average computation time can be seen as the number of tasks per set is increased, the time required still remains very low. The low execution requirements of the ILP implementation comes down to the desire to simply discover if a suitable schedule exists, no optimisation is required. As the number of tasks per set increases, the number of variables required to model the problem increases dramatically. However due to the binary nature of the variables that decide where a task is placed and the lack of a maximization requirement the execution time remains low.

It is also possible to observe this scalability with regard to the number of CPU cores in a system. Figure 4 shows again 99.99% of all task sets tested[3]. It is clear that our ILP solution is scalable both as tasks and CPU cores are added to the system.

During this investigation we do not claim to have the most efficient implementation of the WF heuristic and other aspects of the code could affect the timing results. The results are representative of the real world performance of the solutions and of their performance relative to each other. By increasing the number of tasks per set and cores we have shown that our ILP implementation is scalable, further reinforcing the argument for its use during the development of real world industrial systems.

[2]A small number did not complete within a 24 hour test period. The outliers are omitted as over 80000 task sets were tested per plot while the outliers numbered less than 100.

[3]as before all but a very small number of the outliers completed within 0.11 seconds and a small number did not completed within the 24 test period.

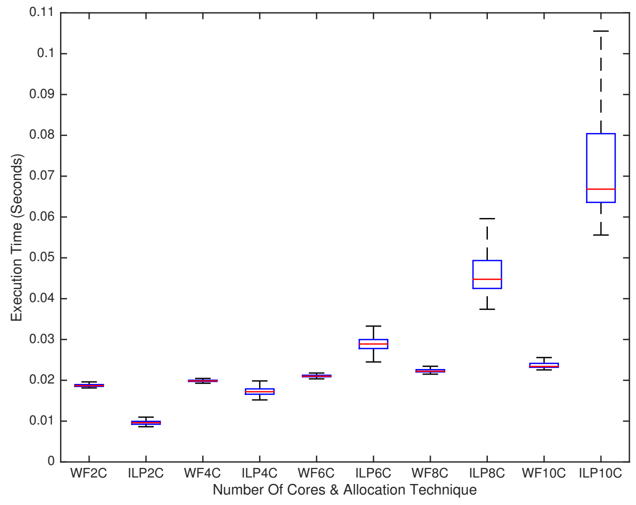

Fig. 4. The increase in computation time as the number of tasks per set is increased.

IV. Conclusions

Throughout this work we addressed the following problems. Firstly we extended the mixed criticality cyclic executive model used in [5] to consider multiple minor cycles per major cycle. This significantly increased the complexity of the allocation problem as now a schedule must be constructed by allocating tasks to the appropriate number of minor cycles, and from there to cores within each minor cycle.

Secondly we showed that when the heuristic based approach WF, is applied to the more complex system model it performs poorly due to the increased complexity of the allocation process. We introduce the notion of using ILP to model the CE system and check for a suitable valid schedule. We show that in the complex case ILP provides significant gains in schedulability over WF. Interestingly in the simple case with a single minor cycle we observed that the heuristics tested (excluding FF) perform very well and manage a level of schedulability close to that provided by ILP. In addition to this, the high MC performance of ILP clearly implies similarly good performance for the non-MC case.

Finally we showed through code timings that our ILP implementation was able to produce a result for a single task set within a very reasonable time frame. In addition to this we investigated the scalability of the solution showing that, although the execution time required did increase as a result of increasing the number of tasks in a set, the time taken was still very reasonable. We showed that for any practical application, an ILP model could be comfortably used to generate a mixed criticality cyclic executive schedule.

The specific ILP tool employed did demonstrate a small number of runs that either took an excessive amount of time to complete or indeed did not completed within a 25 hour period. Those that did complete were mainly unscheduledable. It is therefore a sensible pragmatic approach to deem the task set to be unschedulable if the tool did not obtain a result within 4 seconds.

At the start of this work we posed the question: Is it worth using an optimal solver for CE schedule generation over heuristic based techniques? We have shown that it is worth using, both from the angle of performance and computational efficiency. In addition to this we have shown that the ILP model proposed is scalable allowing it to handle practical system parameters with ease.

Acknowledgements

The authors acknowledges the support and funding provided for this work by BAE Systems, and the ESPRC (UK) via MCC grant (EP/K011626/1). The authors also wish to thank Sanjoy Baruah for insightful discussions on the content of this work.

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] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. *Computers, IEEE Transactions on*, 61(8):1140 –1152, aug. 2012.

[3] S. Baruah and A. Burns. Achieving temporal isolation in multiprocessor mixed-criticality systems. In *WMC*, page 21, 2014.

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

[5] A. Burns, T. Fleming, and S. Baruah. Cyclic executives, multi-core platforms and mixed criticality applications. ECRTS 2015, 2015.

[6] 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.

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

[8] 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.