CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Technical Report

On the Processor Utilisation Bound of the C=D Scheduling Algorithm

José Augusto Santos-Jr
George Lima
Konstantinos Bletsas*

*CISTER Research Center

CISTER-TR-141208

2013/03/13

On the Processor Utilisation Bound of the C=D Scheduling Algorithm

José Augusto Santos-Jr, George Lima, Konstantinos Bletsas*

*CISTER Research Center

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: jamjunior@ufba.br, ksbs@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

Under semi-partitioned multiprocessor scheduling some (or most) tasksare partitioned to the available processors while the rest may migrate between differentprocessors, under a carefully managed scheme. One of the best performingand practical to implement EDF-based semi-partitioned algorithms is C=D splitting.Under this algorithm, each migrating task always executes at the highestpriorityon all but one of the processors that it uses. This arrangement allows forefficient processor utilisation in general, however no utilisation bound had beenpublished so far for this algorithm. We address this situation by deriving the utilisationbound of 13/18 for a variant of C=D with the following constraint: at mostone migrating task may utilise each processor. We also draw additional conclusionsfor the utilisation bound attainable under a C=D task splitting scheme in thegeneral case.

On the Processor Utilisation Bound of the C=D Scheduling Algorithm

J. Augusto Santos-Jr*, George Lima*, and Konstantinos Bletsas+

*Federal University of Bahia, Salvador, Brazil
+CISTER/INESC-TEC Research Centre, ISEP, Porto, Portugal

Abstract. Under semi-partitioned multiprocessor scheduling some (or most) tasks are partitioned to the available processors while the rest may migrate between different processors, under a carefully managed scheme. One of the best performing and practical to implement EDF-based semi-partitioned algorithms is C=D splitting. Under this algorithm, each migrating task always executes at the highest-priority on all but one of the processors that it uses. This arrangement allows for efficient processor utilisation in general, however no utilisation bound had been published so far for this algorithm. We address this situation by deriving the utilisation bound of $\frac{13}{18}$ for a variant of C=D with the following constraint: at most one migrating task may utilise each processor. We also draw additional conclusions for the utilisation bound attainable under a C=D task splitting scheme in the general case.

1 Introduction

Consider the problem of scheduling n sporadic tasks over m identical processors to meet deadlines. Traditionally, scheduling algorithms for this problem were classified as *partitioned* or *global*. Partitioning divides the task set into disjoint subsets each of which is assigned to a respective processor and scheduled there under some uniprocessor algorithm such as Earliest-Deadline-First (EDF) or Fixed Priority Scheduling (FPS). Global scheduling on the other hand, maintains a single run queue for all tasks and, at any given instant, the m highest priority tasks execute, each on one of the m processors. Hence, under global scheduling, tasks may migrate, even halfway through the execution. Partitioning offers simplicity and re-use of techniques from uniprocessor scheduling. The family of global scheduling algorithms dominates partitioning in terms of achievable system utilisation but at the cost of higher scheduling overheads and scalability issues.

The novel *semi-partitioned* scheduling paradigm aims to combine the best aspects of partitioning (efficient implementation and low dispatching overheads) with those of global scheduling (efficient utilisation of available processing capacity). Various semi-partitioned scheduling schemes have been devised. Some are based on fixed-priority scheduling [18] [17] [13] [19] but most are based on EDF [17] [21] [1] [11] [12] [3] [2] [6] [16] [7][8]. The work on semi-partitioning has also influenced the design of novel, unconventional global scheduling schemes [24] [20] [22] [23], especially with regard to the degree of sophistication in the management of task migrations.

The C=D algorithm is an EDF-based semi-partitioned approach which has been shown to have high average-case performance. Migrating tasks are split into pieces; all

but one piece execute as a zero-laxity subtask. Both migrating and non-migrating tasks are scheduled by local EDF, which means that no special implementation requirement is enforced. These characteristics make C=D a very good algorithm from a practical perspective. From a theoretical point of view, though, the least utilisation bound that can be ensured by C=D is not known up to now. In this paper we analyse this problem and provide useful insights into the theoretical aspects of C=D by looking into a new and more restrictive variant of C=D, named Clustered C=D. We provide then a general characterization of the C=D approach in terms of utilization bound; highlight some aspects regarding the task splitting scheme to be used; and establish, as our main result, the fact that the least upper bound on processor utilisation that can be guaranteed by the C=D approach is higher than $\frac{13}{18} = 72.\bar{2}\%$ but cannot exceed 75%.

2 About the C=D scheduling algorithm

Notation We denote by Γ the entire set $\{\tau_1, \ldots, \tau_n\}$ to be scheduled and by Γ_p the set of non-migrating tasks assigned to processor P_p. The term $U(\Gamma_p)$ denotes the utilisation of the task subset Γ_p whereas the utilisation of an individual task τ_i is denoted by u_i.

2.1 Outline of the C=D approach

C=D splitting [9] is a well-performing EDF-based semi-partitioned scheduling approach. Its main characteristic is that each migrating task always executes at the highest-priority on all but one of the processors that it utilises; by comparison, other tasks are scheduled as background workload under EDF on their respective processors. This characteristic also accounts for the name of the approach, since one way of ensuring that a task executes at the highest priority without straying from the semantics of a pure EDF policy is to set its deadline (D) equal to its execution requirement (C). Therefore, each migrating task is modelled as such a zero-laxity ($C = D$) task on the first $k-1$ of the k processors that it is assigned to. The fact that the two scheduling arrangements (assigning to a task the highest priority vs modelling the same task as zero-laxity) are equivalent is formally proven by the following lemma:

Lemma 1. *Let a system S consist of a task τ_0 scheduled at the highest priority on some processor P_p, together with a set Γ_p of implicit-deadline background tasks, which are scheduled under EDF. Let another system S' consist of the same set of tasks $\tau_0 \cup \Gamma_p$ scheduled under pure EDF, wherein τ_0 is assigned a relative deadline $D_0 = C_0$. Then S is schedulable under its respective scheduling policy if and only if S' is schedulable under EDF.*

Proof. Consider the two systems S and S' and the set of all possible legal task arrival patterns. For each such arrival pattern there exist two complementary cases:

- **Case 1:** The respective schedules produced are identical. In this case, either both S and S' meet their deadlines or both miss deadlines.

– **Case 2:** The respective schedules differ. Then, consider the first time instant in $[0, \infty)$ where the two schedules diverge. This can only occur when S' is executing a job by a task $\tau_j \in \Gamma_p$, whose absolute deadline is earlier that that of an active job by τ_0; in comparison, S would be executing τ_0 on the same instant. It then follows that S will miss a deadline (by τ_j) and S' will also miss a deadline (by τ_0).

Hence, we have shown that S and S' miss deadlines for the same subset of legal arrival patterns (which may be null). Hence, S is schedulable if and only if S' is schedulable.

With uniprocessor EDF, which is what the dispatcher on each processor uses, it is possible to have up to one zero-laxity task and still meet deadlines, provided that the execution requirement C of that task is sufficiently small. In practice though [9][4], it has been observed that, almost always, the execution requirement of the zero-laxity task can be made such that the processor is utilised above 90% and the system still remains schedulable. According to the authors of C=D, it was this observation that guided its design.

In terms of implementation, a migratory task is scheduled as a zero-laxity task (or, equivalently, at the highest priority) on each processor that it utilises, except for the last one, whereupon it is scheduled as a regular (non zero-laxity) EDF task. Since the task cannot be preempted when executing at zero laxity, no bookkeeping is needed for tracking its accumulated execution time and averting overruns on the respective processor. Rather, the task migrates to its next processor at the end of the pseudo-deadline associated with each piece. This migration can be set up by a timer. On its last processor, the migrating task is not modelled as zero-laxity; rather, its is scheduled under EDF with an associated relative pseudo-deadline set to D (the true deadline of the task) minus the sum of the respective relative pseudo-deadlines on the previous processors.

2.2 Existing variants

C=D was introduced not as a rigid algorithm but rather as a general approach, which can accommodate different strategies for task assignment and splitting. Therefore, it is not inherently tied to any particular bin-packing scheme – either First-Fit or Best-Fit can be, used for example. However, according to one classification criterion, we distinguish between the following two approaches to task splitting: (i) interleaved bin-packing and splitting or (ii) all bin-packing strictly preceding all task splitting.

The first approach attempts to assign as many tasks as possible on the current processor (starting from the first one) until there is no task which could be assigned there integrally with schedulability preserved, subject to existing assignments. At this stage, some unassigned task τ_i is selected for splitting between the current processor P_p and the next one P_{p+1}. The first "piece" (or subtask) of τ_i is modelled as a zero-laxity task on P_p whose execution requirement is set (according to exact sensitivity analysis) to the maximum value (say C'_i) that preserves the schedulability. The remaining execution requirement of τ_i is modelled as another subtask, with execution requirement $C_i - C'_i$ and relative deadline $D_i - D'_i$, which is added to the pool of unassigned tasks. Subsequently, the bin-packing continues on P_{p+1} until there is need to split again in the same manner. The algorithm succeeds if all tasks are assigned; it fails if it runs out of processors. Note

that it is possible to ensure that the remaining "stub" subtask of τ_i will not have to be split again by assigning it directly to P_{p+1}, prior to any other task, rather than adding to the pool of unassigned tasks.

The second approach, on the other hand, only switches to task splitting once no additional task can be integrally assigned to any processor with schedulability preserved, subject to existing assignments. Remaining unassigned tasks are then split in the manner earlier described, over as many processors as necessary.

These two general approaches can be used with different bin-packing schemes as well as different orderings for the selection of which task to pack or split next. In simulations, these choices are shown to have some effect on average-case performance, depending also on the way that the task parameters are generated. However, in the context of hard-real time scheduling, *a priori* guarantees for the scheduling performance of an algorithm are desirable. Yet, so far, there exists no proven utilisation bound for any variant of C=D, despite empirical evidence of its good performance. This situation motivated this work. In the next section, we will formulate a new variant of C=D, designed so as to facilitate the derivation of its utilisation bound. Subsequently, in Section 4 we are going to prove the respective utilisation bound.

3 The clustered variant of C=D under consideration

In order to find out what utilisation bounds are possible for an algorithm design using a C=D approach with respect to task splitting, we took a step back from the variants already outlined in [9]. Rather, we opted for a simplification of those existing variants, which would make the respective utilisation bound easier to derive – even at the expense of average-case performance.

This simplification consists in allowing at most one migratory task to utilise each processor. In other words, there is at most one piece, by a respective migratory task, on each processor. This piece (subtask) may or may not be zero-laxity. This arrangement divides the processors into disjoint clusters, which share no tasks. Hence we term the approach *Clustered C=D*. One of the motivations for our approach is that, in the context of prior work [15], we already have some results on the schedulability of implicit-deadline tasks scheduled under EDF on one processor under interference from a single higher-priority task, which we can apply.

Outline The high-level pseudocode in Figure 1 defines the clustered approach. Tasks are first ordered by non-increasing T_i. This particular ordering is important, as we will later show, because our derivation of the algorithm's utilisation bound relies on it. Tasks are then assigned one by one, integrally (using First-Fit bin-packing) or, if that fails, by splitting (according to a C=D approach). Each time that a task is split it forms a cluster, consisting of consecutively indexed processors. A variable (initialised to 1) tracks the index of the processor from which the next cluster (if necessary to create) should start. The algorithm can only fail if some task (which previously could not be assigned integrally) cannot be split over the remaining processors (line 8).

Figure 2 depicts an example of a 16-task set assigned onto 6 processors using Clustered C=D. In this example, tasks τ_1 to τ_{13} were successfully assigned integrally. This

```
1. //tasks are indexed by non-increasing T_i
2. int q=1; //stores index of processor to split onwards from
3. for (each task τ_i)
4. {try assigning τ_i integrally to some processor using
    First-Fit bin-packing and an exact schedulability test;

5.  if (τ_i could not be assigned to any processor with
        schedulability preserved, subject to existing assignments)
6.  {re-index processors P_q to P_m by non-decreasing utilisation;

7.   split τ_i in k (as few as possible) pieces using a C=D approach,
     over processors P_q to P_{q+k-1}, using exact schedulability test
     to maximize each zero-laxity piece;

8.   if (we ran out of processors)
9.    return(FAILURE);
10   else
11.   q=q+k;
12.  }
13. }

    //this line is reached only if all tasks were
    //assigned (integrally or by splitting)
14. return(SUCCESS);
```

Fig. 1. Pseudocode for Clustered C=D splitting.

was not the case for τ_{14}, which had to be split. Hence the first cluster $\{P_1,\ P_2,\ P_3\}$ is formed. It happens that the next task τ_{15} can be assigned integrally on some processor. However, the last task (τ_{16}) has to be split (this time, from processor P_4 onwards) and another cluster $\{P_4,\ P_5\}$ is formed. Processors with no migrating tasks can also be thought of as clusters of a single processor. In this example, this is the case of $\{P_6\}$.

Motivation for clustering As hinted earlier, enforcing this clustered arrangement entails a potential performance penalty due to fragmentation. Namely, unlike processors with zero-laxity subtasks, the last processor in each cluster may be underutilised. This is because, although that processor may still have spare processing capacity, by design, this capacity cannot be used for accommodating an additional split task piece. Nevertheless, the clustering approach formulated above can be used as an intermediate step in the derivation of a non-clustered solution which would not suffer from the fragmentation effects described. Not only that, but the derivative non-clustered variant of C=D (which we will next outline) would dominate Clustered C=D, hence also "inheriting" its least schedulable utilisation bound. This is important because, so far, no utilisation bound for the original non-clustered variants of C=D has been proven.

A dominant non-clustered variant of C=D can be obtained from Clustered C=D as a special case, by preventing the algorithm from declaring failure at line 8. Instead, at

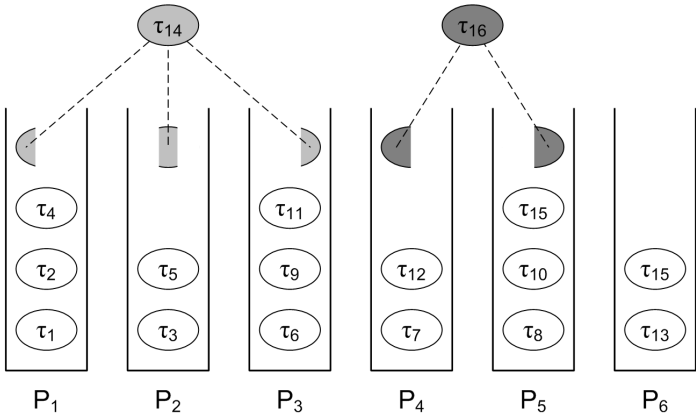

Fig. 2. An example of task assignments under Clustered C=D splitting.

that point the algorithm could allow all processors with spare capacity (i.e. including the last processor of each cluster) to be used, in an attempt to accommodate remaining unassigned tasks. This would undo the clustering – but only if absolutely necessary. Therefore, we can draw the following conclusions:

Lemma 2. *For the non-clustered variant of C=D outlined above it holds that:*

1. *It dominates Clustered C=D.*
2. *Its least upper bound on schedulable utilisation is no less than that of Clustered C=D.*

Proof. By design, given the comments above.

Some interesting observations regarding the pseudocode of Clustered C=D Before proceeding with the derivation of the utilisation bound, for the algorithm described above, it is worth providing some insight into certain aspects of its design. We will do this by studying the behaviour of C=D in different examples, without some of our explicit provisions.

First, it is worth stressing the importance of assigning/splitting tasks in order of non-increasing inter-arrival time.

Lemma 3. *If tasks are not allocated to processors in order of non-increasing inter-arrival time, the least upper bound on processor utilization achieved by the C=D algorithm is not greater than 50%.*

Proof. Consider the implicit-deadline task set $\Gamma = \{\tau_1, \ldots, \tau_{m+1}\}$, with each task ($C$, D, T) having the following attributes:

$$1 \leq i \leq m : \tau_i = \Big(0.5+\varepsilon, \quad 1, 1\Big)$$
$$\tau_{m+1} = \Big((0.5-\varepsilon)m+\varepsilon, m, m\Big)$$

where m is the number of processors and $\varepsilon \to 0^+$. In this case, τ_{m+1} will be the only migrating task, and will be broken up into m zero-laxity subtasks with $C = D = 0.5 - \varepsilon$ and $T = m$ on processors P_1 to P_m. However, this would still leave it with an outstanding execution requirement of ε time units, which cannot be accommodated by any processor. Therefore,

$$\lim_{m\to\infty} \frac{U(\Gamma)}{m} = \frac{1}{2}$$

Observe that both Clustered C=D and the non-clustered variant dominating it would fail, for the above example. This observation led us to enforce the task ordering by non-increasing inter-arrival time, as an integral aspect of Clustered C=D.

Even under this ordering, though, we can see that the least upper bound on processor utilization for C=D is not higher than 75%.

Theorem 1. *The least upper bound on processor utilisation for the C=D algorithm is not higher than $\frac{3}{4}m$ for large values of m.*

Proof. Without loss of generality, assume that m is divisible by 4. We will construct a task set Γ with $n = \frac{3m}{4} + 1$ tasks that could not be dealt with by the C=D scheme. Let Γ be composed of three types of tasks: there are

- m type-1 tasks in the form $(\frac{1}{2} + \varepsilon,\ 1, 1)$;
- $\frac{m}{4}$ type-2 tasks in the form $(\frac{3}{4} - \frac{\varepsilon}{2}, \frac{3}{4}, \frac{3}{4})$;
- and one type-3 task in the form $(\frac{1}{4}, \frac{1}{2}, \frac{1}{2})$;

where ε is a small positive constant. From Lemma 3 we know that unless the tasks are allocated in non-increasing order of their periods, the least upper bound on processor utilization cannot be greater than 50% of m in the general case. Hence, assume a non-increasing order. This means that all type-1 tasks are allocated as non-migrating and the other tasks should be defined as migrating, those of type 2 being allocated first.
When defining the subtasks of the type-2 tasks, one will see that 3 of them should be allocated as having their relative deadlines equal to their execution costs. That is, these three subtasks of each type-2 task always execute with the highest priority (recall Lemma 1). As there are up to two instances (jobs) of these type-2 subtasks interfering in the execution of the non-migrating tasks, their maximum execution cost c must satisfy the following:

$$2c + \frac{1}{2} + \varepsilon = 1 \Rightarrow c = \frac{1 - 2\varepsilon}{4}$$

Now the type-3 task must be allocated. We note that after allocating type-2 tasks, there are $\frac{3m}{4}$ processors where no other task can be allocated and $\frac{m}{4}$ with utilization $\frac{1}{2} + 2\varepsilon$, corresponding to the utilization of the type-1 task and that of the fourth subtask of a type-2 task. We also note that the maximum interference a type-2 task can suffer during its execution cannot exceed $\frac{\varepsilon}{2}$, which is its slack. This means that there are $\frac{m\varepsilon}{8}$ of unallocated processor utilization that can be used but the type-3 task requires 50% of a processor. Computing $U(\Gamma)$,

$$U(\Gamma) = \sum_{1}^{m} \left(\frac{1}{2} + \varepsilon\right) + \sum_{1}^{\frac{m}{4}} \left(\frac{3 - 2\varepsilon}{3}\right) + \frac{1}{2} = \frac{3m}{4} + \frac{2m\varepsilon}{3} + \frac{1}{2}$$

Assuming $\varepsilon \approx 0$, we have that

$$\lim_{m\to\infty} \frac{U(\Gamma)}{m} = \lim_{m\to\infty} \frac{3}{4} + \frac{1}{2m} = \frac{3}{4}$$

The above theorem indicates that if we would be able to prove a least utilisation bound for Clustered C=D which would be close to 75%, then the performance of this algorithm (and its non-clustered dominant variant) would also be close to the theoretical limits inherent to the C=D approach – in other words, the exact utilisation bound, whatever that is.

Another aspect of the pseudocode for Clustered C=D that merits discussion is the re-indexing of processors (performed at line 6). This occurs whenever some task needs to be split and involves all the candidate processors to accommodate it (P_q to P_m). These processors are rearranged then by order of non-decreasing utilisation, before the algorithm proceeds with splitting the task in consideration from P_q (as derived by the re-indexing) onwards, over as many processors as needed. This arrangement, as will be seen later, is relied upon for the derivation of the utilisation bound.

Hence, after having highlighted the motivation behind the design aspects of Clustered C=D, we can now turn to identifying its utilisation bound.

4 Derivation of the utilisation bound of Clustered C=D

In this section, we will prove the utilisation bound of $\frac{13}{18}$ for the clustered variant of C=D above introduced. Given that utilisation bounds are a meaningful performance metric only when the task set τ to be scheduled is implicit-deadline, we henceforth assume that for each task τ_i, it holds that $C_i \leq D_i = T_i$.

4.1 Useful results from the domain of uniprocessor scheduling

The following few results address the schedulability of uniprocessor systems. However, they form the foundations for later proving the utilisation bound of the multiprocessor scheduling scheme in consideration.

Theorem 2. *Let τ_0 be a task scheduled at the highest priority on some processor P_p, together with a set Γ_p of implicit-deadline background tasks, which are scheduled under EDF. Additionally, assume that $T_0 \leq T_j$, $\forall \tau_j \in \Gamma_p$. Then, if*

$$u_0 \leq \frac{1 - U(\Gamma_p)}{1 + U(\Gamma_p)} \tag{1}$$

no deadlines can be missed.

Proof. It is known from Theorem 3 in [15] that, for the case described above, no deadlines can be missed if

$$u_0 \leq \frac{1 - U(\Gamma_p)}{1 + \frac{U(\Gamma_p)}{\left\lfloor \frac{\min_{\tau_j \in \Gamma_p} T_j}{T_0} \right\rfloor}}$$

Since, from the assumption, it holds that $T_0 \leq \min_{\tau_j \in \Gamma_p} T_j$, the above sufficient condition for schedulability can be relaxed to Inequality (1)

Corollary 1. *The utilisation of each zero-laxity subtask assigned to a processor P_p by the clustered C=D algorithm during task splitting is lower-bounded by $\frac{1-U(\Gamma_p)}{1+U(\Gamma_p)}$.*

Proof. Follows from Theorem 2 and Lemma 1 and the fact that the algorithm uses an exact schedulability test to determine the maximum execution requirement by the subtask that preserves the schedulability of Γ_p. Therefore, the right-hand side of Inequality (1) is a lower bound for that utilisation.

4.2 Proof of utilisation bound

At this stage, we can tackle the derivation of the least schedulable utilisation bound for Clustered C=D. For that derivation, we will rely on the above results for uniprocessors, with each zero-laxity subtask of a migrating task corresponding to τ_0 on its respective processor.

Theorem 3. *The total task utilisation $U_{c\ell.}$ accommodated by each cluster formed by the Clustered C=D algorithm is lower-bounded by $\frac{13}{18}k$, where k is the number of processors in that cluster.*

Proof. Let us assume that the cluster in consideration spans processors P_q to P_{q+k-1}. From Corollary 1 and the fact that the split task "exhausts" the available scheduling capacity on the first $k-1$ processors of the cluster (P_q to P_{q+k-2}) (but still needs a final piece on the k^{th} processor to be schedulable), it follows that:

$$u_i > \sum_{p=q}^{q+k-2} \frac{1-U(\Gamma_p)}{1+U(\Gamma_p)} \tag{2}$$

From the fact that $\frac{d}{dx}\left(\frac{1-x}{1+x}\right) < 0$ and $\frac{d^2}{dx}\left(\frac{1-x}{1+x}\right) > 0$ over [0,1], via application of Jensen's inequality [14] to the right-hand side of Inequality (2), we obtain

$$\sum_{p=q}^{q+k-2} \frac{1-U(\Gamma_p)}{1+U(\Gamma_p)} > (k-1)\cdot\frac{1-\bar{U}}{1+\bar{U}} \tag{3}$$

where

$$\bar{U} = \frac{1}{k-1}\sum_{p=q}^{q+k-2} U(\Gamma_p) \tag{4}$$

Combining this with Inequality (2), we obtain

$$u_i > (k-1)\cdot\frac{1-\bar{U}}{1+\bar{U}} \tag{5}$$

Now, let us consider two cases:

– **Case 1:** $k = 2$
Then the cluster consists of two processors P_q and P_{q+1}, over which task τ_i is split. From the bin-packing scheme used, it follows that:

$$\begin{aligned} U(\Gamma_q) + U(\Gamma_{q+1}) &> 1 \\ u_i + U(\Gamma_q) &> 1 \\ u_i + U(\Gamma_{q+1}) &> 1 \end{aligned}$$

Adding these inequalities together and dividing by 2 yields

$$u_i + U(\Gamma_q) + U(\Gamma_{q+1}) \geq \frac{3}{2} > 2 \cdot \frac{13}{18} \Rightarrow U_{c\ell.} > \frac{13}{18} k \tag{6}$$

– **Case 2:** $k \geq 3$
Then, for the total utilisation $U_{c\ell.}$ of the cluster it holds that

$$\begin{aligned} &U_{c\ell.} = u_i + \sum_{p=q}^{q+k-1} U(\Gamma_p) = u_i + \sum_{p=q}^{q+k-2} U(\Gamma_p) + U(\Gamma_{q+k-1}) \overset{(5)}{\Longrightarrow} \\ &U_{c\ell.} > U(\Gamma_{q+k-1}) + (k-1)\bar{U} + (k-1) \cdot \frac{1-\bar{U}}{1+\bar{U}} \Rightarrow \\ &U_{c\ell.} > U(\Gamma_{q+k-1}) + (k-1)\frac{\bar{U}^2+1}{1+\bar{U}} \end{aligned} \tag{7}$$

From the fact that processors P_q to P_m are reindexed in order of non-decreasing utilisation, each time that a task is split (line 6 of the pseudocode) it follows that, at the time that τ_i is split, $U(\Gamma_{q+k-1}) \geq \bar{U}$. Taking advantage of this, in conjunction with (7), we obtain:

$$\begin{aligned} &U_{c\ell.} > \bar{U} + (k-1)\frac{\bar{U}^2+1}{1+\bar{U}} \Rightarrow \\ &\frac{U_{c\ell.}}{k} > \frac{1}{k}\left(\bar{U} + (k-1)\frac{\bar{U}^2+1}{1+\bar{U}}\right) \end{aligned} \tag{8}$$

With some algebraic rewriting, this can be equivalently expressed as

$$\frac{U_{c\ell.}}{k} > \bar{U} + \left(\frac{k-1}{k}\right)\left(\frac{1-\bar{U}}{1+\bar{U}}\right) \tag{9}$$

The quantity $\frac{1-\bar{U}}{1+\bar{U}}$ is positive. The quantity $\frac{k-1}{k}$ is also positive and an increasing function of k. Hence the right-hand side of the above inequality is minimised for the smallest value of k, which is $k = 3$ according to the assumption of the case. Via substitution of this value we obtain

$$\frac{U_{c\ell.}}{k} > \bar{U} + \frac{2}{3}\left(\frac{1-\bar{U}}{1+\bar{U}}\right) \tag{10}$$

From the properties of the bin-packing scheme used, it follows that $\bar{U} > \frac{1}{2}$. Moreover, the right-hand side of Inequality (10) is an increasing function of $\bar{U}$ over $[\frac{1}{2},\ 1]$. From these two observations we obtain that a lower bound for the right-hand side of Inequality 10 can be obtained by substituting $\bar{U} = \frac{1}{2}$. Then, we obtain

$$\frac{U_{c\ell.}}{k} > \frac{13}{18} \Rightarrow U_{c\ell.} > \frac{13}{18}k. \tag{11}$$

Hence in either case the claim holds.

Note that the utilisation of a cluster may still increase, after its formation, via the potential integral assignment of additional tasks to its component processors.

Definition 1. *In the case that Clustered C=D declares failure, the term last candidate cluster denotes the set of processors $\{P_q \dots P_m\}$ upon which the algorithm attempted to split a task immediately prior to declaring failure.*

Intuitively, the algorithm attempted to split the task in consideration over candidate clusters $\{P_q, P_{q+1}\}$, $\{P_q, P_{q+1}, P_{q+2}\}$ and so on until $\{P_q, \dots, P_m\}$ but failed in all cases. Note that, in a trivial case, the last candidate cluster could be $\{P_m\}$.

Lemma 4. *Assume that Clustered C=D declares failure upon attempting to split a task τ_i. Let $U_{\ell last}$ denote the utilisation of the last candidate cluster before the attempt to split τ_i and let k be the number of processors in the last candidate cluster. Then, it holds that $U_{\ell last} + u_i > \frac{13}{18}k$.*

Proof. For $k = 1$ the claim trivially holds. For $k \geq 2$:

From the fact that the algorithm declares failure, we know that $k-1$ zero-laxity subtasks of τ_i were assigned on the $k-1$ first processors in the last candidate cluster. On the final processor, the algorithm tested the schedulability of a final k^{th} subtask, with the remaining execution requirement of τ_i, and failed.

From the fact that EDF is a sustainable scheduling algorithm [5], reducing the execution requirement of that last subtask on the last processor cannot negatively impact on its schedulability. Let us therefore reduce its execution requirement (without changing its deadline or interarrival time) to a value that renders it schedulable on the last processor, together will all other tasks assigned there. From Theorem 3, this means that the cluster would then be utilized above $\frac{13}{18}$ – even if the utilisation of τ_i was discounted. In turn, this means that $U_{\ell last} + u_i > \frac{13}{18}k$.

Theorem 4. *The utilisation bound of Clustered C=D is at least $\frac{13}{18}$.*

Proof. We will prove this by showing that if the algorithm fails to schedule a task set Γ, this means that $U(\Gamma) > \frac{13}{18}m$.

Assume that the algorithm declares failure to split a task τ_i. From Theorem 3 we know that all clusters successfully formed by the algorithm up to that point are utilised above $\frac{13}{18}$. From Lemma 4 we also know that, were τ_i to be assigned to the last candidate cluster (shedulability considerations aside), its resulting utilisation would exceed $\frac{13}{18}$. This means that $U(\{\tau_1, \dots, \tau_i\}) > \frac{13}{18}m$. In turn, since $\{\tau_1, \dots \tau_i\} \subseteq \Gamma$, this implies that $U(\Gamma) > \frac{13}{18}m$.

4.3 Observations regarding the bin-packing scheme

The algorithm in Figure 1 uses First-fit bin-packing but other reasonable bin-packing heuristics can be used instead [10]. For example, Best-Fit could be used instead and all of the reasoning that leads to the derivation of the utilisation bound (and thus the bound itself) would still hold.

In order to characterise what properties a candidate bin-packing scheme should possess in order to be used in the Clustered C=D algorithm, instead of First-Fit, without compromising the proven utilisation bound, let us inspect the proof of Theorem 3. Therein, the only properties resulting from the assignment of non-split tasks that are relied upon are the following:

Property 1: $\frac{1}{2} < \bar{U} \stackrel{\text{def}}{=} \sum_{p=q}^{q+k-2} U(\Gamma_p)$, for the first $k-1$ processors in a cluster $\{P_q, \ldots, P_{q+k-1}\}$.
Property 2: $U(\Gamma_{q+k-1}) \geq \bar{U}$, for the last processor in the cluster.

Upon closer inspection, if Property 1 holds, then Property 2 is ensured by the processor re-indexing (line 6 of the pseudocode), prior to each task splitting. Hence, only Property 1 needs to be safeguarded, when switching to an alternative bin-packing scheme.

One simple way of ensuring that this property holds is by preventing by design a task to be assigned to a processor with no other tasks yet assigned to it *unless* this task could not fit on any of the processors with tasks already assigned to them. Note that the order in which processors with tasks already assigned to them are tried is not relevant for ensuring Property 1; it could even vary for each task or it could even be random.

Preventing a task from being assigned to a processor with no other tasks yet assigned to it unless it could not be assigned to any other processor (with tasks) means that, whenever the need to split a task arises, it holds that

$$U(\Gamma_p) + U(\Gamma_r) > 1$$

where P_p and P_r are the two least utilised from among the first $k-1$ processors of the cluster in consideration. In turn, this means that Property 1 holds.

5 Conclusions

Providing high utilisation guarantees without imposing too many restrictions on implementation has been a challenge for the real-time research community regarding the multiprocessor scheduling problem. In this context, the C=D algorithm is noticeable by its implementation simplicity and good average performance. However, no theoretical bound on system utilisation had been derived so far. In this paper we have looked into this open problem and have discussed some important characteristics of C=D. We have shown that if one does not consider assigning tasks to processors in order of non-increasing task periods, the utilisation bound for the C=D algorithm can be as low as 50%. Furthermore, we have established an interval of $[0.72\bar{2}, 0.75]$ within which such a bound lies. These results bring about important theoretical aspects for a scheduling algorithm known to perform well from a practical perspective.

Acknowledgements
José Augusto Santos-Jr has received financial support by CAPES. This work is part of a research project jointly funded by CNPq and Federal University of Bahia.

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within SMARTS project, ref. FCOMP-01-0124-FEDER-020536.

References

1. James H. Anderson, Vasile Bud, and UmaMaheswari C. Devi. An EDF-based Scheduling Algorithm for Multiprocessor Soft Real-Time Systems. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, pages 199–208, 2005.
2. B. Andersson and K. Bletsas. Sporadic Multiprocessor Scheduling with Few Preemptions. In *Proceedings of the 20th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 243–252, 2008.
3. B. Andersson and E. Tovar. Multiprocessor scheduling with few preemptions. In *Proc. 12th IEEE Int. Conference on Embedded and Real-Time Computing Systems and Applications*, pages 322–334, 2006.
4. Patricia Balbastre, Ismael Ripoll, and Alfons Crespo. Optimal deadline assignment for periodic real-time tasks in dynamic priority systems. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems*, pages 65–74, 2006.
5. Sanjoy K. Baruah and Alan Burns. Sustainable scheduling analysis. In *Proceedigns of the 27th IEEE International Real-Time Systems Symposium (RTSS)*, pages 159–168, 2006.
6. Konstantinos Bletsas and Björn Andersson. Notional processors: an approach for multiprocessor scheduling. In *Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 3–12, 2009.
7. Konstantinos Bletsas and Björn Andersson. Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound. In *Proc. of 30th Real-Time Systems Symposium (RTSS)*, pages 447–456, 2009.
8. Konstantinos Bletsas and Björn Andersson. Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound. *Real-Time Systems*, 47(4):319–355, 2011.
9. A. Burns, R.I. Davis, P. Wang, and F. Zhang. Partitioned EDF Scheduling for Multiprocessors using a C=D Scheme. *Real-Time Systems (published online)*, 48(1):3–33, 2011.
10. E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. In Dorit S. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1997.
11. François Dorin, Patrick Meumeu Yomsi, Joël Goossens, and Pascal Richard. Semi-partitioned hard real-time scheduling with restricted migrations upon identical multiprocessor platforms. CoRR abs/1006.2637, 2010.
12. Frédéric Fauberteau, Serge Midonnet, and Laurent George. Improvement of schedulability bound by task splitting in partitioning scheduling. In *Proceedings of the 1st International Real-Time Scheduling Open Problems Seminar (RTSOPS 2010)*, pages 20–21, 2010.
13. Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. Fixed-priority multiprocessor scheduling with liu and layland's utilization bound. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 165–174, 2010.
14. J. Jensen. Sur les Fonctions Convexes et les Inégalités entre les Valeurs Moyennes. *Acta Mathematica*, 30:175–193, 1906.

15. José Augusto Santos Jr. and George Lima. Sufficient schedulability tests for edf-scheduled real-time systems under interference of a high priority task. In *Proc. of the 2nd Brazilian Symposium on Computer Systems Engineering (SBESC)*, pages 1–6, 2012.
16. S. Kato, N. Yamasaki, and Y. Ishikawa. Semi-partitioned scheduling of sporadic task systems on multiprocessors. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS)*, pages 249–258, 2009.
17. Shinpei Kato and Nobuki Yamasaki. Portioned EDF-based scheduling on multiprocessors. In *Proceedings of the 8th ACM/IEEE International Conference on Embedded Software (EMSOFT)*, pages 139–148, 2008.
18. Shinpei Kato and Nobuki Yamasaki. Portioned static-priority scheduling on multiprocessors. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–12, 2008.
19. Karthik Lakshmanan, Ragunathan Rajkumar, and John P. Lehoczky. Partitioned fixed-priority preemptive scheduling for multi-core processors. In *Proc. of the 21st Euromicro Conf. on Real-Time Systems*, pages 239–248, 2009.
20. Greg Levin, Shelby Funk, Caitlin Sadowski, Ian Pye, and Scott Brandt. DP-FAIR: A Simple Model for Understanding Optimal Multiprocessor Scheduling. In *Proceedings of the 22nd Euromicro Conference on Real-Time Systems (ECRTS 2010)*, pages 3–13, 2010.
21. Ernesto Massa and George Lima. A bandwidth reservation strategy for multiprocessor real-time scheduling. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 175–183, 2010.
22. G. Nelissen, V. Berten, J. Goossens, and D. Milojevic. Reducing preemptions and migrations in real-time multiprocessor scheduling algorithms by releasing the fairness. In *Proceedings of the 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 15–24, 2011.
23. G. Nelissen, V. Berten, V. Nelis, J. Goossens, and D. Milojevic. U-EDF: An unfair but optimal multiprocessor scheduling algorithm for sporadic tasks. In *Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 13–23, 2012.
24. Paul Regnier, George Lima, Ernesto Massa, Greg Levin, and Scott Brandt. RUN: optimal multiprocessor real-time scheduling via reduction to uniprocessor. In *Proceedings of the 32nd Real-Time Systems Symposium*, pages 104–115, 2011.