A lookup-table driven approach to partitioned scheduling

Bipasa Chattopadhyay Sanjoy Baruah

Abstract

The partitioned preemptive EDF scheduling of implicit-deadline sporadic task systems on an identical multiprocessor platform is considered. Lookup tables, at any selected degree of accuracy, are pre-computed for the multiprocessor platform. By using these lookup tables, task partitioning can be performed in time polynomial in the representation of the task system being partitioned. Although the partitioning will not in general be optimal, the degree of deviation from optimality is bounded according to the degree of accuracy selected during the pre-computation of the lookup tables.

1 Introduction

Two different efficiency considerations play a role in determining scheduling strategies for embedded real-time systems. On the one hand, they should be efficient to *implement*; on the other, they should ensure efficient *resource utilization*.

In this paper, we consider the partitioned preemptive EDF scheduling of implicit-deadline sporadic task systems (also known as *Liu & Layland* task systems [5]) on identical multiprocessor platforms. It is widely known (see, e.g., [7, 6]) that such partitioning is equivalent to the bin-packing problem, and is hence highly intractable: NP-hard in the strong sense. Resource-allocation strategies that achieve optimal resource utilization are therefore likely to have very inefficient implementations.

In the search for resource allocation strategies that have efficient implementations, various heuristics for task partitioning have been studied and evaluated (see, e.g. [6]). The heuristics evaluated in such studies are those for which very efficient implementations are easily obtained (such as *First-Fit*, *Best-Fit*, *Worst-Fit*, *First-Fit-Decreasing*, etc.; please see [6] for a description of these heuristics in the context of task partitioning). These studies seek to determine *sufficient schedulability conditions* for these heuristics, and compare different heuristics by comparing their respective sufficient schedulability conditions. These results have proved very useful from the perspective of designing hard-real-time systems; however, they do not provide much insight as to how far removed the resource utilization of these different heuristics are, from optimality.

In other related work, Hochbaum and Shmoys [2] have designed a polynomial-time approximation scheme (PTAS) for the partitioning of implicit-deadline sporadic task systems[1] that behaves as follows. Given any positive constant ϕ, if an optimal algorithm can partition a given task system τ upon m processors each of speed s, then the algorithm in [2] will, in time polynomial in the representation of τ, partition τ upon m processors each of speed $(1 + \phi)s$. This can be thought of as a *resource augmentation* result [3, 4]: the algorithm of [2] can partition, in polynomial time, any task system that can be partitioned upon a given platform by an optimal algorithm, provided it (the algorithm of [2]) is given augmented resources (in terms of faster processors) as compared to the resources available to the optimal algorithm.

This is theoretically an immensely significant result since it allows us to perform task partitioning in polynomial time, to any (constant) desired degree of accuracy. However, the practical significance of this result is severely limited by the fact that the algorithm of [2] has very poor implementation efficiency in practice: the constants in the run-time expression for this algorithm are prohibitively large.

This research. We seek to apply the ideas in [2] to come up with an implementation that is efficient enough to be usable in practice. In brief, our approach is to split the computation needed to implement the algorithm of [2] into two parts: *(i)* a computation-intensive part that is done during the process of assembling the platform upon which the task system is to be implemented; and *(ii)* a far more efficiently-implementable part that is done when attempting to schedule any given sporadic task system upon the platform. The computation-intensive part is done only once when the multiprocessor platform is being synthesized, and the results stored in a lookup table. We envision that this table will be

[1]Actually, the result in [2] was expressed in terms of minimizing the *makespan* of a given finite collection of non-preemptive jobs; however, the makespan minimization problem considered in [2] is easily shown to essentially be equivalent to the partitioning problem we are interested in in this paper.

supplied along with the multiprocessor platform (in much the same manner that complex mathematical functions are sometimes implemented in lookup tables on modern processors). This table is therefore available for the designers of real-time systems when they seek to determine whether particular task systems can be scheduled under partitioned EDF upon this platform or not. Using this table, such a question can be answered very efficiently, in time that is a low-order polynomial in the number of tasks in the system.

2 Task Model

As stated in Section 1, we are considering the partitioned preemptive EDF [5, 1] scheduling of implicit-deadline sporadic task systems [5] on identical multiprocessor platforms. Each implicit-deadline sporadic task is characterized by a worst-case execution time (WCET) parameter and a minimum inter-arrival separation parameter; we use the term *utilization* to denote the ratio of the WCET parameter to the minimum inter-arrival separation parameter. It follows from the results in [5] that a necessary and sufficient condition for the tasks assigned to each processor to be schedulable by EDF is that their utilizations sum to no more than the speed of the processor (assumed here to be equal to one).

3 Constructing the lookup table

We now describe the construction of the lookup table that is to be provided with the multiprocessor platform. Let us suppose that we are given a multiprocessor platform consisting of m unit-speed processors. Recall that this table is constructed only once, at the time that the platform is being put together.

Throughout this section, we will illustrate the construction of the lookup table by means of a running example. Let us suppose that the platform in our example consists of 4 unit-speed processors (i.e., $m = 4$).

§**1. Choosing ϵ.** The computation of the lookup table is governed by a parameter ϵ, which is a positive real number. Informally speaking, the task-assignment strategy is centered upon *rounding up* the actual utilizations of tasks to be of the form $\epsilon \times (1+\epsilon)^k$, for some non-negative integer k.

A design decision must now be made, in the form of choosing a value for ϵ. The smaller the value, the smaller the degree of rounding up that is needed, and the closer to optimal our subsequent task-assignment procedure will be (the exact degree of deviation from optimality is derived later, in Theorem 1). However, the size of the lookup table, and the time required to compute it, also depend on the value of ϵ: the smaller the value, the larger the table-size (and the amount of time needed to compute it).

Config. ID	0.3000	0.3900	0.5070	0.6591	0.8568
A	3	0	0	0	0
B	2	1	0	0	0
C	1	0	1	0	0
D	1	0	0	1	0
E	0	2	0	0	0
F	0	1	1	0	0
G	0	0	0	0	1

Table 1. All the maximal single-processor configurations.

For our running example, let us choose the value 0.3 for the parameter ϵ, i.e., $\epsilon \leftarrow 0.3$. (In reality, we would typically choose a far smaller value, but this is sufficient for purposes of illustration here.)

§**2. Determining the utilization values.** Once the value of ϵ is assigned, the possible utilization values to which we will round up are $\epsilon \times (1+\epsilon)^k$ for all integer $k \geq 0$, up to the upper limit of one.

For our example ($\epsilon = 0.3$), these values are

k	**util. value**
0	0.3000
1	0.3900
2	0.5070
3	0.6591
4	0.8568
5	~~1.114~~

We therefore have 5 distinct utilization values to consider: for $k = 0, 1, 2, 3,$ and 4. (The value of 1.114 for $k = 5$ is too large, as are the values for all $k > 5$.)

§**3. Determining legal single-processor configurations.** With these distinct utilization values as determined above, what are the different ways in which a single processor can be maximally filled? (By *maximally* filled, we mean that adding any additional task to the processor renders it non-schedulable by EDF.) Since there are only finitely many distinct utilization values, this can be determined by exhaustive enumeration: simply try all combinations of distinct utilization values until the processor capacity of one is reached. (Several simple counting and programming techniques can be used to optimize this procedure.)

For our running example with $\epsilon = 0.3$, the maximal single-processor configurations are shown in Table 1. The numbers in the headings for columns 2-6 are the 5 distinct utilization values that we determined in §2 above. Each row corresponds to a different maximal single-processor configuration; it may be verified that the sum of the utilizations in each configuration (i) sums to no more than 1.0, and (ii) is at least 0.7, i.e., adding a task with even the smallest utilization would exceed the processor's capacity. (Consider, for example, the configuration labeled "B": the sum of the utilizations is $2 \times 0.3000 + 1 \times 0.3900$, or 0.9900. For the configuration labeled "D", the sum of the utilizations is $1 \times 0.3000 + 1 \times 0.6591$, or 0.9591.)

§**4. Determining legal multi-processor configurations.** We can use the maximal single-processor configurations determined above to determine maximal configurations for a collection of m processors. Since there are only finitely many maximal single-processor configurations, this, too, can be done using exhaustive enumeration: simply try all $m-$combinations of maximal single-processor configurations. (As in Step §3 above, simple counting and programming techniques can be used to optimize this procedure.) These maximal configurations are stored in a lookup table that is provided along with the m-processor platform ,and which is used (as discussed in Section 4 below) for partitioning specific task systems upon the platform.

For our example 4-processor platform with $\epsilon = 0.3$, it turns out that there are 140 maximal configurations. Although this is too many to enumerate in this document, we depict a few in Table 2 in the format that they will appear in the lookup table. The numbers in the headings for columns 1-5 are the 5 distinct utilizations; the sixth column lists the 4 maximal single-processor configurations, named according to the configuration ID's of Table 1, that give rise to this particular maximal 4-processor configuration.

4 Task assignment

Once the lookup table enumerating the maximal m-processor configurations has been obtained, we can use this table to efficiently determine whether any implicit-deadline sporadic task system can be partitioned on this platform or not. We now describe the partitioning algorithm for doing so.

Let τ denote a collection of n implicit-deadline sporadic tasks to be partitioned among the (unit-capacity) processors in the m-processor platform. Let u_i denote the utilization of the i'th task.

The task assignment algorithm first attempts to assign all tasks with utilization $\geq \epsilon/(1+\epsilon)$, in Steps 1 and 2 below. Once this has been completed, tasks with utilization $< \epsilon/(1+\epsilon)$ are considered, in Steps 3 and 4.

1. For each task with utilization $\geq \epsilon/(1+\epsilon)$, *round up* its utilization (if necessary) so that it is equal to $\epsilon \times (1+\epsilon)^k$ for some non-negative integer k. Observe that such rounding up inflates the utilization of a task by at most a factor $(1+\epsilon)$: the ratio of the rounded-up utilization to the original utilization of any task is $\leq (1+\epsilon)$.

2. Now all the tasks with (modified) utilization $\geq \epsilon/(1+\epsilon)$ have their utilizations equal to one of the distinct values that were considered during the table-generation step. Determine whether this collection of these modified-utilization tasks with utilization $\geq \epsilon/(1+\epsilon)$ can be accommodated in one of the maximal m-processor configurations that had been identified during the pre-processing phase.
   - If the answer here is "no," then report failure: we are unable to partition τ among the m processors.
   - If the answer is "yes," however, then a viable partitioning has been found: assign the tasks according to the maximal m-processor configuration.

3. It remains to assign the tasks with utilization $< \epsilon/(1+\epsilon)$. Assign each to any processor upon which it will "fit;" i.e., any processor on which the sum of the (original — i.e., unmodified) utilizations of the tasks assigned to the processor would not exceed one if this task were assigned to that processor.

4. If all the tasks with utilization $< \epsilon/(1+\epsilon)$ cannot be assigned in this manner, then report failure: we are unable to partition τ among the m processors. Otherwise, all the tasks have been assigned and we report success.

Properties. It is straightforward to observe that if this task-assignment algorithm succeeds in assigning all the tasks to processors, then the resulting assignment is indeed a correct one: the sum of the utilizations of the tasks assigned to any particular processor is no larger than one, and hence each processor is successfully scheduled by EDF.

What if the algorithm *fails* to assign all the tasks to the processors?

- Suppose that the algorithm reports failure when attempting to assign only the tasks with utilization $\geq \epsilon/(1+\epsilon)$ (Step 2 above). Since each such task has its utilization inflated by a factor $< (1+\epsilon)$, it must be the case that all such (original — i.e., unmodified-utilization) tasks cannot be scheduled by an optimal algorithm on a platform comprised of m processors each of computing capacity $1/(1+\epsilon)$.

- Suppose that the algorithm reports failure when attempting to assign the tasks with utilization $< \epsilon/(1+\epsilon)$ (Step 4 above). This would imply that the sum of the utilizations of the tasks already assigned to each processor is $> (1-\epsilon/(1+\epsilon))$. Therefore the total utilization of τ exceeds $m(1-\epsilon/(1+\epsilon)) = m(1/(1+\epsilon))$, and

0.3000	0.3900	0.5070	0.6591	0.8568	**From single-proc. configurations**
3	2	1	2	0	[D E D C]
3	4	2	0	0	[F F E A]
0	3	3	0	1	[F F F G]
4	1	1	1	1	[G D C B]
4	0	1	3	0	[D D D C]

Table 2. Some example maximal 4-processor configurations.

τ cannot consequently be feasible on m processors of speed $1/(1+\epsilon)$.

We have thus shown the following:

Theorem 1 *Any task system that can be partitioned on m processors of speed $(1/(1+\epsilon)$ by an optimal partitioning algorithm can be partitioned on m unit-speed processors by our algorithm.* ■

Returning to our example ($m = 4$ processors, $\epsilon = 0.3$), let us consider a task system τ comprised of tasks with the following utilizations (listed here in non-decreasing order):

$$\frac{1}{5}, \frac{1}{5}, \frac{1}{3}, \frac{7}{20}, \frac{9}{25}, \frac{2}{5}, \frac{1}{2}, \frac{1}{2}, \frac{3}{4}.$$

Noting that $\epsilon/(1+\epsilon) = .3/1.3 \approx 0.2308$, we note that the first two tasks have utilization $< \epsilon/(1+\epsilon)$ and are hence not to be considered during the first steps of the partitioning algorithm.

We round the remaining utilizations up:

$$\frac{1}{5}, \frac{1}{5}, 0.3900, 0.3900, 0.3900, 0.5070, 0.5070, 0.5070, 0.8568.$$

Using Table 2, we notice that the rounded-up utilizations here correspond to the configuration listed on the third row: 0 3 3 0 1, obtained from the single-processor configurations [F F F G] of Table 1.

Accordingly, we assign the tasks with utilization $\geq \epsilon/(1+\epsilon)$ to the 4 processors as specified in configurations F, F, F, and G respectively:

F: 1/3, 2/5. (Remaining capacity $= 1 - 0.7333$ = 0.2667)

F: 7/20, 1/2. (Remaining capacity $= 1 - 0.85$ = 0.15)

F: 9/25, 1/2. (Remaining capacity $= 1 - 0.86$ = 0.14)

G: 3/4. (Remaining capacity $= 1 - 0.7500$ = 0.25)

It remains to assign the two tasks with utilization $< \epsilon/(1+\epsilon)$: the ones with utilization $1/5$ each. These first can be accommodated in the first and last processors, yielding the following mapping:

F: 1/3, 2/5, 1/5. (Remaining capacity = 0.0667)

F: 7/20, 1/2. (Remaining capacity = 0.15)

F: 9/25, 1/2. (Remaining capacity = 0.14)

G: 3/4, 1.5. (Remaining capacity = 0.05)

5 Summary

The PTAS of [2] does not find much use in partitioned multiprocessor scheduling due to its large run-time. We are exploring the possibility of breaking the needed computation into two distinct parts: a very computation-intensive part that must be done only once per platform, with the results stored in a lookup table for subsequent use, and a more efficiently-implementable part that essentially consists of table lookup and some simple additional processing. Initial results are promising, but it remains to be sees what degrees of accuracy (as reflected in the value of the parameter ϵ) and for what numbers of processors (m, this technique will scale to.

References

[1] M. Dertouzos. Control robotics : the procedural control of physical processors. In Proceedings of the IFIP Congress, pages 807–813, 1974.

[2] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. Journal of the ACM, 34(1):144–162, January 1987.

[3] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In 36th Annual Symposium on Foundations of Computer Science (FOCS'95), pages 214–223, Los Alamitos, October 1995. IEEE Computer Society Press.

[4] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. Journal of the ACM, 37(4):617–643, 2000.

[5] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM, 20(1):46–61, 1973.

[6] J. M. Lopez, J. L. Diaz, and D. F. Garcia. Utilization bounds for EDF scheduling on real-time multiprocessor systems. Real-Time Systems: The International Journal of Time-Critical Computing, 28(1):39–68, 2004.

[7] J. M. Lopez, M. Garcia, J. L. Diaz, and D. F. Garcia. Worst-case utilization bound for EDF scheduling in real-time multiprocessor systems. In Proceedings of the EuroMicro Conference on Real-Time Systems, pages 25–34, Stockholm, Sweden, June 2000. IEEE Computer Society Press.