Using Processor Speed-up to Control Preemption Related Costs

Abhilash Thekkilakattil, Radu Dobrin and Sasikumar Punnekkat
Mälardalen Real-Time Research Center, Mälardalen University, Sweden
{abhilash.thekkilakattil, radu.dobrin, sasikumar.punnekkat}@mdh.se

I. Related Work

Preemptive real-time schedulers are associated with preemption related overheads, and their effects are challenging to analyze because they typically vary with the point of preemption e.g., cache related preemption delays, and even with the state of the physical process that the real-time system is controlling. Moreover, preemptive scheduling typically requires the use of resource access protocols [1] to enable mutual exclusion, in cases where tasks communicate through shared resources. These resource access protocols, though predictable, introduce schedulability overheads in the system, as well as may lead to pessimistic assumptions in the schedulability analysis. Even though preemptive scheduling schemes are used in a large number of applications, mostly due to its ability to achieve high processor utilization, the detrimental impact of preemptions is widely recognized in the community [2] [3] [4]. The preemption related costs includes the context-switch overhead [5] and to manipulate the task queues [3], as well as the indirect cost of cache-related preemption delays [6]. Bui et al [2] observed a worst case increment in task execution time of upto 33% on a PowerPC MPC7410 with a 2 MB two way associative L2 cache, due to the cache related preemption delays. The worst case temporal overhead due to cache related preemption delays were found to be as high as $655\mu S$ for a single preemption. The rate monotonic algorithm (RM) was shown to introduce a higher number of preemptions than earliest deadline first algorithm (EDF) by Buttazzo [7], increasing the overheads in the system and hence reducing the benefits of its simple runtime implementation.

The applicability of a non-preemptive scheduling scheme, on the other hand, is limited to only a small fraction of the feasible task sets [8] due to its inability to fully utilize the computational resources in most of the cases [9]. Task sets scheduled by non-preemptive schemes can be deemed as unschedulable even at arbitrarily low utilizations [10]. On the other hand, the major benefits of using non-preemptive scheduling is the absence of preemption related costs and that it does not require any resource access protocols, as mutual exclusion is guaranteed by the scheduler. However, the problem of finding a non-preemptive schedule for a given task set is either NP-hard or infeasible for most task sets [11].

In order to take advantage of the benefits of both preemptive and non-preemptive scheduling paradigms, various limited preemption scheduling models were proposed, a detailed survey of which can be found in [12]. A benefit of using these approaches is that non-preemptive regions can be enforced within the task executions. The use of a limited preemption technique helps in bounding the preemption related costs and reducing it, i.e., it basically ensures that a preemption occurs only when absolutely necessary and/or at an optimal point with the least cost. It also ensures that, whenever possible, critical sections can be executed entirely non-preemptively. However, the limited preemption approach does not provide for a fine grained ability to control the preemption behavior of real-time tasks e.g., to guarantee a user specified bound on the number of preemptions per task. To guarantee mutual exclusion during critical section execution, it is essential that the non-preemptive region of each task is larger than its largest critical section. Similarly, to minimize preemption related costs, the length of the non-preemptive region of any task must be no less than the length of the task execution between any two consecutive optimal preemption points, where the cost of a preemption is the least. If the length of the non-preemptive region does not guarantee a specified preemption behavior, a preemptive or non-preemptive schedule may not be feasible. For example, if the preemptions are not possible at optimal preemption points, it may increase the task execution time by 33% [2], potentially causing deadline misses. Augmenting the limited preemption scheduling schemes with the flexibility for specifying a specified non-preemption behavior can further enhance its applicability in modern real-time systems to control the preemption related costs.

Of the several methods that have been proposed to reduce the number of preemptions in real-time scheduling, Preemption Threshold Scheduling (PTS) for FPS was first introduced in the ThreadX operating system by Lamie [13]. This scheme was later formalized by [14], by providing an accurate analysis and also proposing an optimal algorithm to calculate the preemption thresholds. Jeffay et al [11] derived the sufficient and necessary conditions for non-preemptive feasibility of periodic and sporadic tasks. They also showed that EDF is an optimal algorithm for scheduling tasks non-preemptively under a work conserving paradigm. Baruah [9] proposed the limited preemption model, which was subsequently named as Floating Non-Preemptive Region model (f-NPR model), in which they proposed an algorithm to calculate the length of the longest possible non-preemptive execution of a task in a sporadic task system. Later, [15] evaluated the approach for randomly generated task sets, showing its effectiveness. In an earlier work, they had extended the f-NPR scheduling scheme to FPS [16], where they found an upper bound on the length of the largest possible non-preemptive execution of a task under FPS and presented extensive simulation results. Later, [17] presented a method to optimally place preemption points within the task code, assuming a fixed preemption overhead, and presented extensive simulation results for both EDF and FPS. The evaluation results showed that the limited preemption models are more effective than non-preemptive and fully preemptive scheduling schemes with preemption costs, in successfully scheduling task sets. [18] proposed a method to improve the schedulabilility of FPS by executing the last portion of the tasks in a non-preemptive fashion, as long as possible. Dobrin and Fohler [19] proposed a method to identify preemption offline and to control the number of preemptions by changing the task parameters, such as the priority, to avoid these conditions required for a preemption. An extensive survey of the various limited preemption methods can be found in [12] and a detailed comparison can be found in [10].

The energy consumption in a processor can be modeled by the relation $P = CV^2F$, where P is the power consumed by the processor, V is the applied voltage, C is the effective capacitance and F is the operating frequency [20]. This relation indicates that using a slower processor would decrease the energy consumption. However, when using Dynamic Voltage Scaling (DVS) [20] for energy efficiency, due to an increase in task execution times, the number of preemptions increases significantly. [21] found that a frequency switch can be done in less than $140\mu S$, which amounts to just one fifth of the cost of a single preemption making CPU frequency scaling a promising approach towards controlling preemption behavior in real-time systems. In [22] [23] [24], methods were proposed to control the preemption behavior of sporadic and periodic tasks scheduled by FPS using CPU frequency scaling. In [25], an approach to combine PTS with DVS to enable energy efficient scheduling was presented. However, it does not provide for controlling the preemption behavior of the schedule.

All the above limited preemption approaches still require the support of a preemptive scheduler on top of which additional support mechanisms are required for their implementation e.g., dual priority scheduler for PTS [14] and timers for limited preemption models [9] [15]. None of the above methods are able to completely eliminate preemptions, even though they are able to greatly reduce the preemption overheads. These methods also do not provide for controlling the preemption related costs in the schedule e.g., ensuring no more than a user desired number of preemptions per task or guaranteeing that a preemption is always possible at an optimal preemption point in the task. In this context, we examine, and prove the possibility of, using a faster processor to achieve effective preemption related cost control in the schedule. Controlling the preemption related costs provides a system designer with the ability to perform trade-offs between, e.g., energy- preemption overhead trade-offs.

II. Translating Preemption Cost Control Requirements to Non-Preemption Requirements

The key idea behind the approach is that the system level requirements of meeting specified bounds for various preemption related costs can be translated into a set of task level non-preemption requirements, i.e., a lower bound on the length of the non-preemptive regions for each task.

In the following, we outline the first step in our approach i.e., the translation of preemption cost control requirements to specified non-preemption requirements for each task that will, for example, lower the overall preemption related overheads. Our aim is to find the optimal processor speed that will guarantee that the length of the non-preemptive region per task at that speed is greater than or equal to the non-preemption requirement.

A. *Control the Bound on the Number of Preemptions*

The method to control the bound on the number of preemptions has been presented in the main paper.

B. *Enable Preemptions at Optimal Preemption Points*

As mentioned earlier, the preemption related costs also depend on the points at which the preemptions occur. If a preemption cannot be avoided, it is preferable to have it at points where the cost of a preemption is the least, i.e., optimal preemption points. The possibility of enforcing preemptions only at these optimal preemption points depends on the length of the non-preemptive region on a processor of a given speed S. Remember that $q^S_{i,j}$ denotes the length of execution of τ_i up to its j^{th} optimal preemption point on a speed S processor. Hence, the non-preemption requirement for a task τ_i is given by the largest interval between any two consecutive optimal preemption points of τ_i when it executes on a processor at a speed S:

$$L^S_i = \max_{1 \leq j < m} (q^S_{i,j+1} - q^S_{i,j}, q^S_{i,1})$$

Consequently, our goal is to find the processor speed, S, that satisfies:

$$Q^S_i \geq L^S_i = \max_{1 \leq j < m} (q^S_{i,j+1} - q^S_{i,j}, q^S_{i,1}) \qquad (1)$$

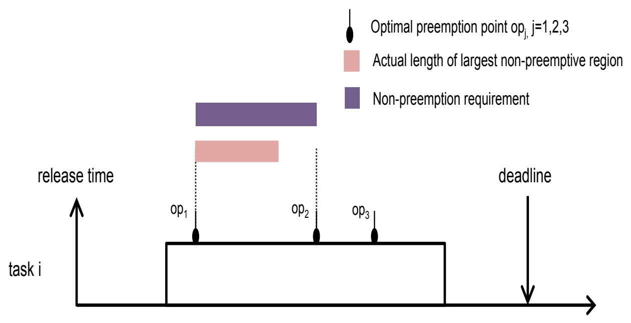
Optimal preemption point op$_j$, j=1,2,3
Actual length of largest non-preemptive region
Non-preemption requirement
release time
deadline
op$_1$
op$_2$
op$_3$
task i

Figure 1. Non-preemption requirement to enable preemptions only at optimal preemption points

Thus if the processor executes at speed S, any premption on τ_i can be deferred to the closest optimal preemption point.

An illustrative example is given in figure 1 where the non-preemption requirement of a task that guarantees preemptions only at optimal preemption points, is greater than its largest non-preemptive execution. In the figure, the actual length of the largest non-preemptive regions is calculated using the result by [9]. If a high priority task is released immediately after the optimal preemption point op_1, the preemption cannot be deferred to the next optimal preemption point op_2 as the largest possible non-preemptive region for τ_i is less than the length of the task execution between op_1 and op_2. In our example, for τ_i, the largest length of the task execution between any two optimal preemption point is given by the length of the task execution between op_1 and op_2. Hence, to ensure that the preemptions can always occur at one of the optimal preemption point, the length of the largest possible non-preemptive region should be at least as large as the length of the task execution between op_1 and op_2. Only then can we ensure that any preemption that may potentially occur after op_1 can be guaranteed to be deferred to the optimal preemption point op_2.

C. Execute Critical Sections within Non-Preemptive Regions

An attractive feature of the limited preemption scheduling paradigm is that it has the potential to enable access to shared resources without the need for synchronization mechanisms, by executing the critical sections within non-preemptive regions. However, this is not always possible, e.g., in case the length of the non-preemptive region Q_i^S, for a task τ_i is shorter than its largest critical section CS_i^S on a processor of speed S. This issue, on the other hand, can be solved by finding an adequate processor speed S that enables $Q_i^S \geq L_i^S$. The processor speed that guarantees this is given by the speed S that will satisfy the relation:

$$Q_i^S \geq L_i^S = CS_i^S$$

An illustrative example is given by figure 2 where the non-preemption requirement of a task that guarantees the non-preemptive execution of its critical sections is greater the actual length of its largest non-preemptive region.

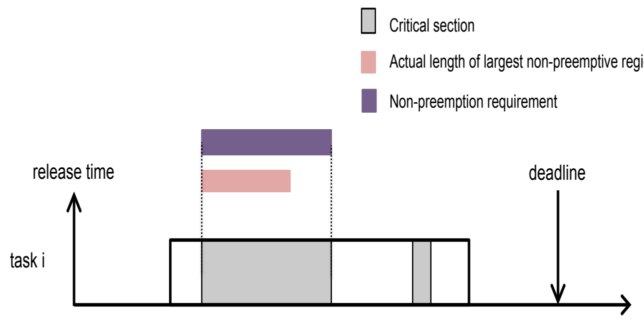
Critical section
Actual length of largest non-preemptive region
Non-preemption requirement
release time
deadline
task i

Figure 2. Non-preemption requirement to always enable critical section execution inside non-preemptive regions

In figure 2, the actual length of the largest non-preemptive region is calculated using the result by [9]. If a high priority task is released immediately after the start of the first critical section, then the preemption cannot be deferred to a point after the critical section, requiring the use of synchronization protocols. Hence the non-preemption requirement should be atleast as large as the largest critical section to ensure the execution of the critical section within a non-preemptive region.

III. Relaxing the Assumption of Linear Speed-up

In this section, we discuss how we can relax some of the assumptions made in this paper. Specifically, we consider relaxing the assumptions of linear speed-up and negligible preemption related overheads at optimal preemption points.

So far in this paper, we have considered the possibility of a linear speed-up. While such an assumption allows us to derive interesting theoretical results on the resource augmentation bounds for preemption control, it might not be a reasonable assumption from a practical point of view because of the effects of memory wall [26]. In this sub-section, we consider relaxing the assumption of a linear speed-up by considering the execution time model proposed by [27]. Marinoni et al [27] assumed that the execution time of a task consists of two parts- one that scales linearly with the processor frequency and one that does not scale with the processor frequency. Following their notation, let us define ϕ as the percentage of the execution time for every τ_i, that scales with the processor frequency.

Let us see how the theorem IV.I changes when we consider our new assumption.

Theorem III.1. *The processor speed S_i that guarantees the feasibility of a non-preemption requirement L_i for a task τ_i is given by,*

$$S_i = \max_{D_1 \le t < D_i} \left\{ \frac{\phi \sum_{j=1}^{n} DBF_j^1(t)}{t - L_i - (1-\phi)\sum_{j=1}^{n} DBF_j^1(t)} \right\}$$

Proof: The length of the non-preemptive region for τ_i at speed 1 is given by [9],

$$Q_i^1 = \min_{D_1 \le t < D_i} \left\{ t - \sum_{j=1}^{n} DBF_j^1(t) \right\}$$

$$\Rightarrow Q_i^1 \le t - \sum_{j=1}^{n} DBF_j^1(t), \forall t, D_1 \le t < D_i$$

Our aim is to find the processor speed S_i that guarantees the feasibility of a non-preemption requirement L_i. We know that, of the total demand bound in any interval t, only ϕ percentage scales with the processor frequency. Thus,

$$L_i \le t - \left\{ \frac{\phi \sum_{j=1}^{n} DBF_j^1(t)}{S_i} + (1-\phi)\sum_{j=1}^{n} DBF_j^1(t) \right\}, \forall t, D_1 \le t < D_i$$

Hence,

$$S_i L_i \le S_i t - \left\{ \phi \sum_{j=1}^{n} DBF_j^1(t) + S_i(1-\phi)\sum_{j=1}^{n} DBF_j^1(t) \right\}, \forall t, D_1 \le t < D_i$$

Solving for S_i, we get,

$$S_i \ge \left\{ \frac{\phi \sum_{j=1}^{n} DBF_j^1(t)}{t - L_i - (1-\phi)\sum_{j=1}^{n} DBF_j^1(t)} \right\}, \forall t, D_1 \le t < D_i$$

i.e.,

$$S_i = \max_{D_1 \le t < D_i} \left\{ \frac{\phi \sum_{j=1}^{n} DBF_j^1(t)}{t - L_i - (1-\phi)\sum_{j=1}^{n} DBF_j^1(t)} \right\}$$

■

Lemma III.1. *The speed S_i that guarantees the feasibility of a non-preemption requirement L_i for any task τ_i, during a time interval t is upper-bounded by,*

$$S_i \le \frac{\phi y}{\phi y - 1}$$

where, $y = \frac{t}{L_i}$, $\forall t \in [D_1, D_i)$.

Proof: We know from theorem III.1 that,

$$S_i = \max_{D_1 \le t < D_i} \left\{ \frac{\phi \sum_{j=1}^{n} DBF_j^1(t)}{t - L_i - (1-\phi)\sum_{j=1}^{n} DBF_j^1(t)} \right\}$$

Since we have assumed that the task set is feasible, the upper-bound on the value of $\sum_{j=1}^{n} DBF_j^1(t)$ is t. Hence,

$$S_i \leq \left\{ \frac{\phi t}{t - L_i - (1-\phi)t} \right\}$$

which gives,

$$S_i \leq \frac{t}{\phi t - L_i}$$

Finally, substituting $y = \frac{t}{L_i}$,

$$S_i \leq \frac{\phi y}{\phi y - 1}$$

■

We can now use similar reasoning as in lemma IV.2, IV.3 and IV.4 to derive upper-bounds on the processor speed-up required that provides the required non-preemption guarantees. We however leave more details to a future work, since in this paper we focus on the fundamental question of how the preemption behavior changes with the processor speed.

References

[1] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *The IEEE Transactions on Computers*, September 1990.

[2] B. D. Bui, M. Caccamo, L. Sha, and J. Martinez, "Impact of cache partitioning on multi-tasking real time embedded systems," in *The IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2008.

[3] A. Burns, K. Tindell, and A. Wellings, "Effective analysis for engineering real-time fixed priority schedulers," *The IEEE Transactions on Software Engineering*, 1995.

[4] H. Ramaprasad and F. Mueller, "Tightening the bounds on feasible preemptions," in *The ACM Transactions on Embedded Computing Systems*, 2008.

[5] D. I. Katcher, H. Arakawa, and J. K. Strosnider, "Engineering and analysis of fixed priority schedulers," *The IEEE Transactions on Software Engineering*, 1993.

[6] C.-G. Lee, J. Hahn, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim, "Analysis of cache-related preemption delay in fixed-priority preemptive scheduling," *The IEEE Transactions on Computers*, 1998.

[7] G. Buttazzo, "Rate monotonic vs. EDF: Judgment day," in *The ACM International Conference on Embedded Software*, 2003.

[8] S. K. Baruah, L. E. Rosier, and R. R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Systems*, 1990.

[9] S. Baruah, "The limited-preemption uniprocessor scheduling of sporadic task systems," in *The Euromicro Conference on Real-Time Systems*, 2005.

[10] G. Yao, G. Buttazzo, and M. Bertogna, "Comparitive evaluation of limited preemptive methods," in *The International Conference on Emerging Technologies and Factory Automation*, 2010.

[11] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of periodic and sporadic tasks," in *The IEEE International Real-time Systems Symposium*, 1991.

[12] G. Buttazzo, M. Bertogna, and G. Yao, "Limited preemptive scheduling for real-time systems: A survey," *The IEEE Transactions on Industrial Informatics*, 2012.

[13] W. Lamie, "Preemption threshold," *Whitepaper, Accessed: 29 May 2012*, 1997. [Online]. Available: http://rtos.com/articles/18833

[14] Y. Wang and M. Saksena, "Scheduling fixed-priority tasks with preemption threshold," in *The International Conference on Real-Time Computing Systems and Applications*, 1999.

[15] M. Bertogna and S. Baruah, "Limited preemption EDF scheduling of sporadic task systems," *The IEEE Transactions on Industrial Informatics*, November 2010.

[16] G. Yao, G. Buttazzo, and M. Bertogna, "Bounding the maximum length of non-preemptive regions under fixed priority scheduling," in *The IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2009.

[17] M. Bertogna, G. Buttazzo, M. Marinoni, G. Yao, F. Esposito, and M. Caccamo, “Preemption points placement for sporadic task sets,” in *The Euromicro Conference on Real-Time Systems*, 2010.

[18] M. Bertogna, G. Buttazzo, and G. Yao, “Improving feasibility of fixed priority tasks using non-preemptive regions,” in *The IEEE Real-Time Systems Symposium*, 2011.

[19] R. Dobrin and G. Fohler, “Reducing the number of preemptions in fixed priority scheduling,” in *The Euromicro Conference on Real-time Systems*, 2004.

[20] P. Pillai and K. G. Shin, “Real-time dynamic voltage scaling for low-power embedded operating systems,” in *The ACM symposium on Operating systems principles*, 2001.

[21] J. Pouwelse, K. Langendoen, and H. Sips, “Dynamic voltage scaling on a low-power microprocessor,” in *The international conference on Mobile computing and networking*, 2001.

[22] A. Thekkilakattil, R. Dobrin, and S. Punnekkat, “Probabilistic preemption control using frequency scaling for sporadic real-time tasks,” in *The International Symposium on Industrial Embedded Systems*, 2012.

[23] ——, “Preemption control using CPU frequency scaling in real-time systems,” in *The International Conference on Control Systsems and Computer Science*, 2011.

[24] A. Thekkilakattil, A. S. Pillai, R. Dobrin, and S. Punnekkat, “Reducing the number of preemptions in real-time systems scheduling by CPU frequency scaling,” in *The International Conference on Real-Time and Network Systems*, 2010.

[25] R. Jejurikar and R. K. Gupta, “Integrating processor slowdown and preemption threshold scheduling for energy efficiency in real time embedded systems,” in *The IEEE Real-Time Computing Systems and Applications*, 2004.

[26] W. A. Wulf and S. A. McKee, “Hitting the memory wall: implications of the obvious,” *SIGARCH Computer Architecture News*, vol. 23, no. 1, March 1995.

[27] M. Marinoni and G. Buttazzo, “Elastic dvs management in processors with discrete voltage/frequency modes,” *IEEE Transactions on Industrial Informatics*, February 2007.