Reducing the Number of Preemptions in P-FRP *

Chaitanya Belwal and Albert M.K. Cheng
Department of Computer Science,
University of Houston, TX, USA
{cbelwal, cheng}@cs.uh.edu

Abstract

A purely functional model of computation, called P-FRP, offers a new paradigm for building reactive software. By allowing static priority assignment, P-FRP guarantees that, when a higher priority task is released, the system will immediately preempt any lower-priority tasks that may be executing at that time. To maintain the atomicity of execution, changes made to the system by an aborted task are rolled back. This gives P-FRP a unique execution paradigm as it does not fit into the strict definitions of the classical preemptive or non-preemptive models of execution. Since preemptions are accompanied by cache related overheads which affect schedulability, real-time researchers have been involved in developing techniques for reducing the number of preemptions. However, most of these studies deal with the preemptive model, where the notion of critical instant and optimality of the rate-monotonic priority assignment hold true. In previous works, we have shown that the results for critical instant and optimality of the rate-monotonic assignment do not hold true in P-FRP. Hence, it is unknown if available results for the preemptive model can be adopted 'as is' for P-FRP. In P-FRP, the additional 'abort' cost incurred during a preemption, makes preemption reduction an important tool to enhance the schedulability of the task set. In this paper, we evaluate existing techniques and present new ones for reducing the number preemptions in P-FRP.

1. Introduction

Reactive programming is a paradigm where program variables dependant on external input are automatically updated when any change in the input occur. For example consider a function: $f(x,y) \leftarrow x + y: x,y \in$ *external input.* In reactive programming, any change in the value of either x or y will result in $f(x,y)$ being re-evaluated by the mechanism of the reactive programming system. Hence the value of $f(x,y)$ will continuously change over time as input values are changed. This programming style is used for writing software in reactive systems many of which are real-time in nature. Reactive programming has been implemented in an imperative language [6], as an Object-Oriented model [22] and in the Functional Programming Model [29].

Functional programming is based on the mathematical principles of lambda calculus (λ-calculus). A formal definition of functional programming was given by Backus, by introducing the FP programming language [5]. Since then many flavors of Functional languages have been developed including OCaml [9], Haskell [15] and Scheme [25]. Functional languages are also called declarative languages as they are mainly concerned with the definition of computational logic rather than the control flow. Since they use functional evaluation, the execution of a functional program does not require variables and other mutable data that represents the state of the execution. (In practice however, for ease of use some shared and mutable data is allowed at the discretion of the programmer. Only *purely functional* languages do not allow any use of mutable data).

Using the Functional paradigm in reactive programming offers the advantage of an inherently safe programming environment for writing software. Functional Reactive Programming has been implemented for applications like robotics [23] and embedded controllers [18]. FRP [29] was developed as a domain specific language on Haskell. A subset of FRP was used to develop RT-FRP [27] where the space and time cost of execution is bounded. The resource boundness of RT-FRP makes it well suited for embedded systems. A compilation strategy is proposed to convert a modified RT-FRP language into efficient imperative code [28]. The language of this new system called E-FRP, is for use in embedded real-time controllers, and has been tested on a small microcontroller driven robot. The event execution in E-FRP follows the simple *first-in-first-out* order. This may cause these events to miss their deadline leading to potentially catastrophic results in real-time systems. To overcome this, a priority based E-FRP system called priority-based FRP (P-FRP) [17] has been developed. P-FRP retains the exact computational semantics of E-FRP but uses fixed priority scheduling to pre-assign a priority number to every event before execution. Hence, P-FRP offers guarantees for both responsiveness and resource boundness, making it ideal for memory and power constrained systems. However, a requirement in the functional programming model is that the state of the system cannot be changed, and no function can have side effects. To maintain this guarantee of stateless execution, the functional programming paradigm requires the execution of a function to be atomic in nature. To comply with this requirement, as well as allow preemption of lower priority events, P-FRP implements a transactional model of execution. Using only a copy of the state during event processing and atomically committing these changes at the end of the event handler (or *task*), a multi-version commit model of execution is implemented. This ensures that handling an event is an "all or nothing" proposition, and ensures the atomicity of handling an event.

* This work is supported in part by U.S. National Science Foundation under Award no. 0720856

In P-FRP, a preempted task is aborted and has to restart when no higher priority tasks are available for execution. The time spent in the aborted execution of a low priority task is termed the *abort* cost. As an illustration, consider two tasks τ_1, τ_2 with execution times 4 and 3, respectively, with τ_2 having a higher priority. Let the arrival periods of τ_1, τ_2 be 8 and 12, and release times be 0 and 3 respectively. The execution of this task set in P-FRP is shown in *Figure 1(a)*. Since τ_1 is released first, it starts execution but is preempted by the release of τ_2 at time 3. After τ_2 completes execution at time 6, τ_2 commences execution but is unable to complete before the release of its second job at time 9. The time spent on the aborted execution of τ_1 in the interval [0,3) is the abort cost induced on τ_1 by τ_2.

In this example, the task can be made schedulable by changing the release offset of τ_2 to 0. With this change, τ_2 starts execution first, and both tasks will be able to complete processing in their feasibility interval of [0, 24) (*Figure 1(b)*). The tasks will also be schedulable if the priorities of τ_2 and τ_1 are reversed (*Figure 1(c)*), or the preemption of τ_1 is deferred (*Figure 1(d)*), allowing the 1st job of τ_1 to complete execution. Clearly, there are several techniques that can be used to reduce the number of preemptions in P-FRP. In this paper, we will present offline methods that analyze the task set and determine changes that can be made in task attributes or the scheduling policy that reduce the number of preemptions.

1.1 Related Work

The transactional model of execution of P-FRP is similar, but not same, as the execution model found in lock-free and transactional memory systems. From previous work [1,2,3,13,17,24] on these models it is easy to deduce that the temporal characteristics of transactional models is quite different from the standard model of preemptive and non – preemptive execution that have been the main focus of real-time research. Several researchers have looked into methods that improve the schedulability by reducing the number of preemptions, and many approaches have been proposed for the preemptive model. In this model, the state of any preemptable task has to be stored temporarily in a cache. Read / write operations to the cache can be expensive, and minimizing such costs has been the primary motivation in reducing the number of preemptions.

Wang and Saksena [30] have applied preemption threshold in fixed priority scheduling. Defining a preemption threshold allows the scheduler to allow preemption for only those tasks whose priority level is below the threshold. Hence, preemption threshold allows the system to implement a hybrid preemptive / non-preemptive scheduling policy. The ThreadX [12] RTOS is an example of a commercial system allowing definition of preemption threshold. In [19], the authors propose a scheme to avoid preemptions in processors using support from dynamic voltage scaling. A selected voltage scale up is used to hasten the execution of lower priority tasks. However, this scheme is limited in the sense that it is useful in cases where the processor is not capable of running at full speed for longer durations. In normal cases, the processor is run at full speed and energy optimization algorithms are used to slow down the tasks without effecting real-time schedulability. The authors propose another approach of delayed preemption, where the start of a higher priority tasks is delayed to allow a lower priority to complete execution.

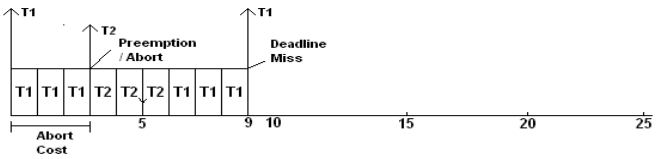

Figure 1(a): Task execution showing the abort cost induced on τ_1 by τ_2, causing τ_1 to have a deadline miss at time 9. T1 and T2 represent tasks τ_1 and τ_2 respectively.

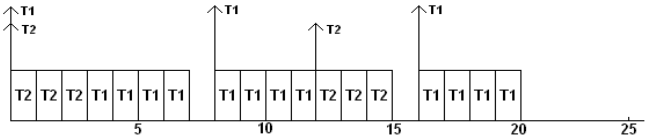

Figure 1(b): If τ_2 is released at time 0, both tasks are schedulable

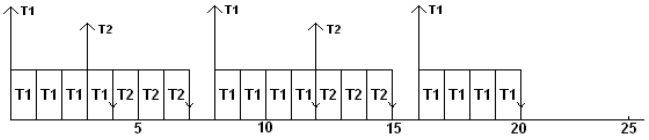

Figure 1(c): If priorities are reversed, such that τ_1 has the higher priority then both tasks are schedulable

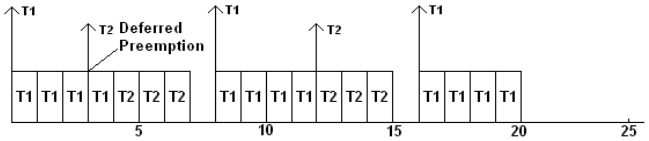

Figure 1(d): If the preemption of τ_1 by the 1st job of τ_1 is deferred, both tasks are schedulable

Dobrin and Fohler [10] have presented several techniques to reduce the number of preemptions by optimizing task attributes using Integer Linear Programming (ILP). However, as shown in [10] reduction in preemption does come with a cost of inflexibility in task execution. In [7], response time analysis under deferred preemption / cooperative scheduling has been presented. In a cooperative scheduling policy tasks are broken up into several non-preemptable regions, boundaries of which are classified as preemption points. It has been unknown so far, if any of these approaches can be adopted for P-FRP.

In previous work for P-FRP, response time analysis and priority assignment strategies have been discussed. In [3], it has been shown that the rate-monotonic priority assignment is not optimal and determining an optimal priority assignment for P-FRP is being done in ongoing work.

2. Basic Concepts and Execution Model of P-FRP

In this section we introduce the basic concepts and the notation used to denote these concepts in the rest of the paper. In addition, we review the P-FRP execution model and assumptions made in this study.

2.1 Basic Concepts

Essential concepts for P-FRP are tasks and their associated priority, their associated time period and the concept of arrival rate and their processing time; the concept of a trace and task jobs therein. All tasks parameters are assumed to be

known *a priori*. The notation and formal definitions for these concepts as well as a few others used in the paper are as follows:

- Let **task set** $\Gamma_n = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of n periodic tasks, and $\tau_{i,m}$ represent the m^{th} job of task τ_i
- The **priority** of $\tau_k \in \Gamma_n$ is pr_k. If $pr_k > pr_j$ then τ_k has a higher priority than τ_j
- T_k is the **arrival time period** between two successive jobs of τ_k and $r_k = 1 / T_k$ is the **arrival rate** of τ_k
- C_k is the **worst-case execution time** for τ_k
- $R_{k,m}$ represents the **release time** of the m^{th} job of τ_k
- Φ_k represents the **release offset** which is the release time of the first job of τ_k. Or, $\Phi_k = R_{k,1}$. Hence, $R_{k,m} = \Phi_k + (m-1)\cdot T_k$
- $[t_1,t_2)$ represent the **time window** for analyzing gaps, such that: $\forall t \in [t_1,t_2),\ t_1 \le t < t_2 \wedge t_1 \ne t_2$
- D_k is the **relative deadline** of τ_k. If some job of τ_k is released at time $R_{k,m}$ then τ_k should complete processing by time $R_{k,m} + D_k$, otherwise τ_k will have a **deadline miss.** In this paper, $D_k = T_k$
- A **feasibility interval** is the time interval $[t_H,\ t_H + H)$ such that if all tasks are schedulable in $[t_H,\ t_H + H)$ then the tasks will also be schedulable in the time interval $[0, Z)$: $Z \to \infty$. H is the length of the feasibility interval and t_H is its start time
- **Interference** on τ_k is the action, where the processing of τ_k is interrupted by the release of a higher priority task. In P-FRP, an interference forces τ_k to abort and re-start later

2.2 Execution Model and Assumptions

For this study all tasks are assumed to execute in a uni-processor system and have no precedence constraints. When job of a higher priority task τ_i is released it can immediately preempt a lower priority task, and changes made by the lower priority task are rolled back. The lower priority task will be restarted when the higher priority task has completed processing. When some task is released it enters a processing queue Q, which is arranged by priority order such that all arriving higher priority tasks are moved to the head of the queue. The length of the queue is bounded, and no two instances of the same task can be present in the queue at the same time. This requires a task to complete processing before the release of its next job. To maintain this requirement, we assume a hard real-time system with task deadline equal to the time period between jobs. Hence,

$\forall k \in \Gamma_n,\ D_k = T_k$.

Once a task τ_i enters Q two situations are possible. If a task of lower priority than i is being processed, it will be immediately preempted and τ_i will start processing. If a task of higher priority than τ_i is being processed then τ_i will wait in the Q and start processing only after the higher priority task has completed. An exception to the immediate preemption is made during *copy* and *restore* operations which is explained in the following paragraph.

In P-FRP, when a task starts processing it creates a 'scratch' state, which is a *copy* of the current state of the system. Changes made during the processing of this task are maintained inside such a state. When the task has completed, the 'scratch' state is *restored* into the final state in an atomic operation. Therefore during the restoration and copy operations the task being executed cannot be preempted by higher priority tasks. Both copy and restore operations involve simple copy of variables, which takes only a fraction of the execution time of a task. Hence, any blocking which copy and restore operations can induce on a higher priority task, have been ignored in this study. For reducing preemptions, the only task attributes that can be modified are the release offsets and priority assignment. The task arrival rates and worst-case execution times are assumed constant.

In response time analysis for fixed-priority scheduling, a *critical-instant* of release is assumed. Critical instant is the time, at which task releases lead to the worst-case response time (WCRT) [20] of the task being analyzed. In their seminal work, Liu and Layland [20] showed that in fixed-priority scheduling for the preemptive model, the critical-instant for a lower priority task τ_i occurs when it is released at the same time as all higher priority tasks. As shown in [2], for P-FRP, a *synchronous* release of τ_i and higher priority tasks is not guaranteed to result in the WCRT of τ_i. Methods to determine WCRT in P-FRP are discussed in [2].

We now look at two methods that can be used to reduce the number of preemptions in P-FRP. First, we look at the task attributes that can be changed, and secondly we look at modifications to the scheduler which can reduce the number of preemptions.

3. Modifications in Task Attributes

Dobrin and Fohler [10], have identified three conditions, to define a preemption. They offer solutions to avoid each condition, effectively avoiding the preemption. While methods for eliminating each condition in P-FRP are being developed in ongoing work, we present a brief overview of each technique.

3.1 Priority Reassignment for Individual Jobs

In [10], each job of a task is individually analyzed for preemption. A preemption can occur if some higher priority task interferes with the execution of a lower priority task. As in [10], we will reverse the priority order for only those individual jobs which interfere with each other. By building priority inequalities for all interfering jobs, an ILP formulation is derived, which is then solved to determine jobs whose priorities have to be modified. However, a side-effect of making this change, is bifurcating jobs whose priority order has been changed into individual tasks themselves. Such new tasks are referred to as artifacts in [10].

3.2 Release Offset of Tasks

Preemptions can be avoided by changing the release offsets of higher priority tasks. This has been illustrated in *figure 1(b)*. However, changing the release offset of a task to avoid preemption of one job changes the release times of all jobs of the same task. This can possibly lead to additional preemptions of future jobs of the same or different task, thus affecting the schedulability of the task set. Hence, after changing the offset for a task, a schedulability analysis in the feasibility interval of the task set has to be done.

3.3 Release Times of Individual Jobs

Consider, $\Gamma_2 = \{\tau_i, \tau_j\}$, with $pr_i > pr_j$. If the finish time of the $\tau_{j,p}$ (denoted as $finish(\tau_{j,p})$) is more than $R_{i,q}$, then τ_i will preempt τ_j. Or, a preemption will occur when:

$$finish(\tau_{j,p}) > R_{i,q}.$$

In [10], it is shown that if the release time of $\tau_{i,q}$ is changed such that :

$$R_{i,q} = finish(\tau_{j,p}) - C_i$$

then, $\tau_{i,q}$ will not preempt $\tau_{j,p}$. A preemption dependency tree is used to determine those jobs whose release times can be changed to make them schedulable.

4. Modifications in Scheduler

Preemptions can be avoided by making necessary modification in the preemption policy of the scheduler. We will look at two approaches that can be used for P-FRP.

4.1 Preemption Threshold

Preemption threshold allows users to define a priority number, such that only those tasks having a priority lower than the threshold are preempted. Tasks with priority equal to, or more than the threshold continue execution in a non-preemptive way. Response time analysis for tasks with a preemption threshold has been done in [30]. While the work in [30] assigns a global preemption threshold that remains static as long as the task is running, we will evaluate if the threshold can be changed for specific jobs of the task. Since, preemption threshold can effect the schedulability of a task set, the threshold number has to be set after careful analysis.

4.2 Deferred Preemptions

In the deferred preemption policy that we will implement, the scheduler will defer preemption if the lower priority task has executed for a predefined number of steps. With this scheduling policy, a maximum bound on abort costs will be defined. Hence, every task will be preemptive only up to a certain point in its execution, after which the task with become non-preemptive. Implementing this policy will affect the schedulability and responsiveness of the tasks, impact of which will be analyzed in ongoing work.

5. Conclusions and Future Work

We have presented a brief overview of techniques that will be adopted to reduce the number of preemptions in P-FRP. Most of these techniques are similar to the ones presented in [7,10,30], but will be modified to work in P-FRP execution model. In ongoing work, algorithms and formal methods to implement these techniques are being developed. Experimental evaluations using synthetic task sets of all presented methods will also be done.

References

1. J. H. Anderson, S. Ramamurthy, K. Jeffay. "Real-time computing with Lock-free Shared Objects". *ACM Transactions on Comp.Sys. 5(6), pp.388-395*, 1997
2. C. Belwal, A.M.K. Cheng. "Determining Actual Response Time in P-FRP", *Technical Report: UH-CS-10-05, Dept. Of Computer Science, University of Houston*, 2010
3. C. Belwal, A.M.K. Cheng. "On Priority Assignment in P-FRP", *RTAS'10 WiP Session*, 2010
4. C. Belwal, A.M.K. Cheng, "On the Feasibility Interval for P-FRP". *Manuscript under review, http://www2.cs.uh.edu/~cbelwal/FeasibilityInterval_PFRP.pdf*, 2010
5. J. Backus. "Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs". *Communications of the ACM (Volume 21 Issue 8), pp. 613 –641*, 1978
6. F.Boussinot. "Reactive C: an extension of C to program reactive systems". *SoftwarePractice & Experience Vol. 21 Issue 4, pp. 401–428* , 1991
7. R.J.Bril, J.J. Lukkien, W.F.Verhaegh. "Worst-Case Response Time Analysis of Real-Time Tasks under Fixed-Priority Scheduling with Deferred Preemption Revisited". *ECRTS. pp.269-279*, 2007
8. J. Byun, A. Burns, A. Wellings. "A Worst-Case Behavior Analysis for Hard Real-time transactions". *Workshop on Real-time Databases*, 1996
9. The Caml Language, *http://caml.inria.fr*
10. R. Dobrin, G. Fohler. "Reducing the Number of Preemptions in Fixed Priority Scheduling". *ECRTS*, 2004
11. C. Elliott, P. Hudak. "Functional reactive animation". *ICFP'97, pp. 263-273*, 1997
12. ExpressLogic, *http://www.rtos.com*
13. S.F. Fahmy, B. Ravindran, E.D. Jensen. "Response time analysis of software transactional memory-based distributed real-time systems", *ACM SAC Operating Systems*, 2009
14. S. Gopalakrishnan, G.M. Parulkar. "Bringing real-time scheduling theory andpractice closer for multimedia computing". *ACM Sigmetrics Performance Evaluation Review, 24(1), pp. 1 - 12*, 1996
15. Haskell, *http://www.haskell.org*
16. M. Herlihy, J.E.B. Moss. "Transactional memory: architectural support for lock-free data structures". *ACM SIGARCH Computer Architecture New (Col. 21, Issue 2),pp. 289-300*, 1993
17. R. Kaiabachev, W. Taha, A. Zhu. "E-FRP with Priorities". *EMSOFT'07 , pp. 221-230* , 2007
18. R. Kieburtz. Realtime Reactive Programming for Embedded Controllers. *Available from author's home page*, 2001
19. W. Kim, J. Kim, S. L. Min. "Preemption-aware dynamic voltage scaling in hard real-time systems". *ISLPED, pp. 393–398, 2004*
20. C. L. Liu, L. W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". *Journal of the ACM (Volume 20 Issue 1), pp. 46 - 61*, 1973
21. J. Manson, J. Baker, A. Cunei, S. Jagannathan, M. Prochazka, B. Xin, J. Vitek. "Preemptable Atomic Regions for Real-Time Java". *RTSS'05, pp.62-71*, 2005
22. J. Nordlander, M. P. Jones, M. Carlsson, R. Kieburtz, A. Black. Reactive Objects, *Fifth IEEE International Symposium on Object-Oriented Real Time Distributed Computing*, 2002
23. J. Peterson, G. D. Hager, P. Hudak. "A Language for Declarative Robotic Programming". *ICRA'99, IEEE*, 1999
24. J. Ras, A. Cheng. "Response Time Analysis for the Abort-and-Restart Task Handlers of the Priority-Based Functional Reactive Programming (P-FRP) Paradigm". *RTCSA'09*, 2009
25. Scheme, *http://groups.csail.mit.edu/mac/projects/scheme/*
26. L. Sha, R. Rajkumar, J. P. Lehoczky. "Priority Inheritance Protocols: An approach to Real Time Synchronization". *Transactions on Computers Volume 39, Issue 9, pp.1175 – 1185*, 1990
27. Z. Wan, W. Taha, P. Hudak. "Real - time FRP". *ICFP'01, pp. 146-156, ACM Press* ,2001
28. Z. Wan, W. Taha, P. Hudak. "Event driven FRP". *PADL'02*, 2002
29. Z. Wan, P. Hudak. "Functional reactive programming from first principles". *ACM SIGPLAN Conference on Programming Language Design and Implementation, pp.242-252*,2000
30. Y. Wang, M. Saksena. "Scheduling fixed-priority tasks with preemption threshold". *RTCSA, pp. 328–335*, 1999