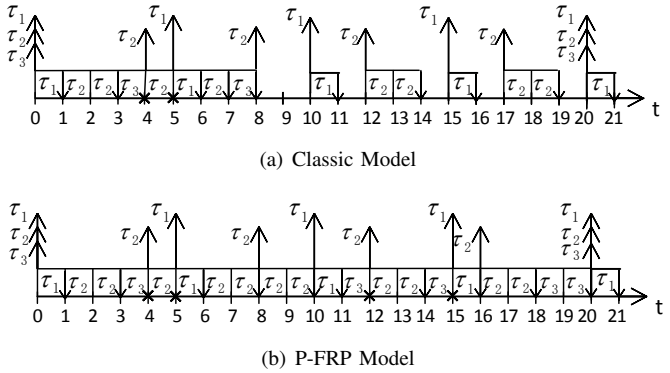
Memory-aware Response Time Analysis for P-FRP Tasks

Xingliang Zou, Albert M. K. Cheng
Department of Computer Science
University of Houston
Houston, TX, 77004, USA
Email: xzou@uh.edu, cheng@cs.uh.edu

***Abstract*—Functional Reactive Programming (FRP) is playing and potentially going to play a more important role in real-time systems. Priority-based (preemptive) FRP (P-FRP), a variant of FRP with more real-time characteristics, demands more research in its scheduling and timing analysis. In a P-FRP system, similar to a classic preemptive system, a higher-priority task can preempt a lower-priority one and make the latter aborted. The lower-priority task will restart after higher-priority tasks complete their execution. Unlike the classic preemptive model, when a task aborts, all the changes made by the task are discarded (abort and restart). In previous studies, the value of Worst Case Execution Time (WCET) of a task is used for all its restarted tasks. However, in practice the restarted tasks likely consume less time than the WCET when considering the memory effect such as cache-hit in loading code and data. In this work, we use different task execution time for restarted tasks when conducting schedulability and response time analysis for P-FRP tasks.**



(a) Classic Model

(b) P-FRP Model

Fig. 1: Schedules of fixed priority task set
($C_1 = 1, C_2 = 2, C_3 = 2, T_1 = 5, T_2 = 4, T_3 = 20$)

I. Introduction

There are two distinct types of programming paradigms in computer programming: imperative and functional. Functional Programming has a distinct difference from imperative programming in that it is immune to side-effects caused by using states and mutable data. Since the formal system of λ-calculus (lambda-calculus) was first devised by Church [1] and Kleene [11], many functional programming languages have been invented: LISP, Ocaml, Haskell, Scheme, Erlang, F#, Atom, Scala and so on. Haskell and Erlang have been studied and commercially developed intensively. Scala is recently adopted by companies such as Linkedin, Twitter and Walmart [7]. Functional Reactive Programming (FRP) [16] is a framework for constructing reactive applications using the building blocks of functional programming.

In real-time systems, the correctness of a program is measured by its logical output as well as its ability to complete within certain time limits. FRP is demonstrated to be effective in modeling and building reactive systems such as graphics, robotic and vision applications. However, another significant feature of real-time systems, priority, is not considered in FRP. To address this problem, the P-FRP [10] model has been proposed as a variant of the FRP model. P-FRP maintains both the type-safety and the state-less execution paradigm of FRP, and supports priorities assigned to different tasks while not requiring the use of synchronization mechanisms between tasks in the system. It has the potential to transform the building of more and more complicated Cyber Physical Systems (CPSs). Christoffersen and Cheng [8] presented an impact of P-FRP in building controllers in automobile anti-lock brake systems.

In order to maintain the state-less paradigm of the FRP model, unlike the classic preemptive scheduling (shortened to classic model) (Fig.1.(a)), P-FRP uses an Abort and Restart (Fig.1.(b)) semantics where if a lower-priority task is interrupted by a higher-priority task, it has to restart from the beginning when it is resumed. Here we consider a typical task life cycle without being interrupted (*cold started* task): (1) code is loaded from hard drive and data is loaded from external memory; (2) computation is done by processor(s); (3) results are committed to external memory. In the P-FRP model, the time spent in phase (2) and (3) is wasted when a task is aborted, however, since the existence of memory hierarchy, the time spent in phase (1) can be less when a task is restarted, for example, the task code is still in cache and does not need to be read from slow external memory again. This memory effect is not considered in previous studies of P-FRP systems. In this paper, we present our preliminary memory-aware P-FRP task response time analysis and experimental results.

II. System Model and Related Works

We consider a hard real-time uniprocessor system with hierarchical memory components. n-task system Γ_n =

*The work is supported in part by the United States National Science Foundation (NSF) under awards No. 0720856 and No. 1219082.

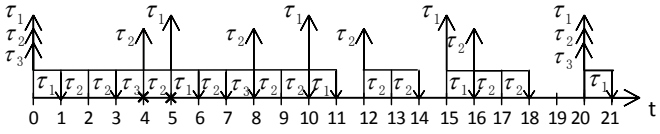
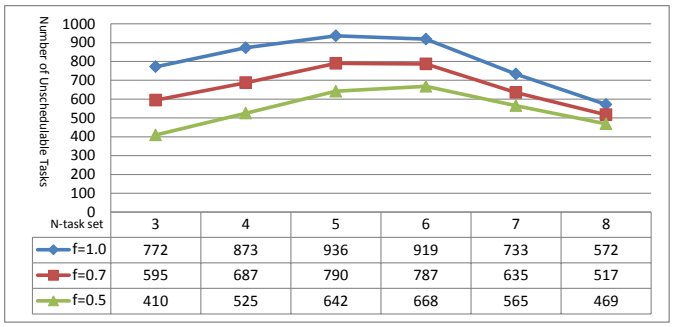


Fig. 2: Memory-aware P-FRP scheduling of fixed priority task set ($C_1^1 = 1, C_1^2 = 1; C_2^1 = 2, C_2^2 = 1; C_3^1 = 2, C_3^2 = 1; T_1 = 5, T_2 = 4, T_3 = 20$)



N-task set	3	4	5	6	7	8
f=1.0	772	873	936	919	733	572
f=0.7	595	687	790	787	635	517
f=0.5	410	525	642	668	565	469

Fig. 3: Number of Unschedulable Task Sets

$\{\tau_1, \tau_2, ..., \tau_n\}$ of periodic tasks with fixed priorities is scheduled in the P-FRP abort and restart model. Task τ_i is assigned a unique fixed priority i $(1 \leq i \leq n)$, where 1 is the highest priority and n is the lowest. An instance or invocation of a periodic task is called a job. J_i^k refers to the k-th job of τ_i. Each periodic task τ_i is characterized by a constant arrival time period T_i between two successive jobs of the task, a relative deadline $D_i (D_i \leq T_i)$, and *two* computation times: C_i^1 and C_i^2, the execution time of the task in *cold started* and *restarted* modes respectively. The Response Time (RT) of a job is the time between the release of a job and its completion. The response time of a task in a given priority assignment is the largest response time among those of all its jobs. A task is referred to as schedulable according to a given scheduling policy if its response time under the given scheduling policy is less than or equal to its deadline. A task set is referred to as schedulable according to a given scheduling policy if all of the tasks in the task set are schedulable under the given scheduling policy.

Jiang *et al* presented their research on P-FRP task schedulability analysis in [13][14] to find tighter feasibility intervals. Belwal and Cheng have shown in [3] that RM is not optimal in P-FRP systems with synchronous release and it is even unknown if there exists an optimal one other than an exhaustive test over all possible priority assignment algorithms. Belwal and Cheng [2] presented a utilization-based analysis that the current schedulability condition only holds true with the utilization bound of $1/n$ under certain restrictions on periods and release scenarios. Wong *et al* [5] conducted research on other priority assignment algorithms: Utilisation Monotonic (UM), Execution-time Monotonic (EM), and a combination of UM and EM, Execution-time-toward-Utilisation Monotonic (EUM) priority assignment algorithm. By comparing with an Exhaustive Search schema, they confirmed that none of RM, DM, EM or EUM is optimal for the P-FRP model. Zhou *et al* [9] presented their research on WCRT and schedulability analysis for real-time software transactional memory-lazy conflict detection for P-FRP tasks. Wong *et al* [6] proposed the Deferred Abort model to reduce the number of preemptions in scheduling P-FRP tasks. Zou *et al* [12] proposed a non-work-conserving Deferred Start model to eliminate preemptions in scheduling P-FRP tasks. However, none of these researches considers the different execution time of *cold* started tasks and restarted tasks. On the other hand, Kazemi and Cheng [15] studied the P-FRP task execution time on a scratchpad memory-based platform, and showed that the task execution time changes because of the memory hierarchy.

III. OUR WORK

A. Memory-aware P-FRP model

In Fig.1, we schedule three tasks with a given priority assignment under the classic model and the P-FRP model. If the execution time of a task in *cold* start and restart cases are different, we will have a memory-aware P-FRP model scheduling as Fig.2 shows.

In Fig.2, the second job of τ_2, J_2^2, is preempted at time point 5 since it requires 2 time units of *cold start execution time* (C_2^1) and executes only 1 time unit when the second job of the highest priority task τ_1 is scheduled to execution. And when J_2^2 is resumed at time point 6, it needs only 1 time unit of *restart execution time* (C_2^2) and hence finishes execution at time point 7. For the job J_2^3 that arrives at time point 8, it requires 2 time units of *cold start execution time* and finishes execution at time point 10. The similar scheduling applies to task τ_3. We can see, under the same priority assignment and scheduling policy, compared to Fig.1(b) of P-FRP scheduling, in time interval of [0, 20) the number of preemptions is reduced from 4 to 2, the CPU idle time unit is increased from 0 to 4. Also, the response time of τ_2 is reduced from 4 to 3, and the response time of τ_3 is reduced from 20 to 8. Thus the memory-aware P-FRP scheduling saves CPU time and potentially is able to schedule more tasks.

B. Experiment and Result

The experiments are designed and conducted on a desktop computer with a CPU of i3-4130 3.4GHz, 8 GB memory and *Ubuntu 14.04.3 LTS 64-bit Desktop* operating system. We generate task sets and run the P-FRP scheduling with and without considering memory effect. The task sets we generated have 3, 4, ..., 8 tasks respectively; the periods are randomly generated in range of $[15, 75]$; the total utilization of a task set is 0.6. We use the *UUniFast* algorithm [4] to generate n $(n = 3, 4, .., 8)$ utilization factors U_i $(1 \leq i \leq n)$ in a descending order such that the total utilization $U = \sum_{i=1}^{n} U_i$ equal to the given value, 0.6 in this experiment. We then shuffle those U_i to a random order for the consideration of generalization. The computation time of each task is computed as $C_i^1 = U_i * T_i$, $C_i^2 = C_i^1 * f$, where f is a factor that shows the difference of task execution time in *cold* start and restart. f is 0.7 and 0.5 in our experiments. As comparison, we also run the task sets in original P-FRP model, which is equivalent to *f=1.0*. For each *n*-task set, we generate 1000 task sets.

Fig.3 shows the number of unschedulable task sets with different f for the n-task sets ($n = 3, 4, .., 8$) we generated. The case of *f=0.7* has 22.9%, 21.3%, 15.6%, 14.4%, 13.4% and 9.6% less unschedulable task sets compared to the original P-FRP scheduling. The case of *f=0.5* has 46.9%, 39.9%, 31.4%, 27.3%, 22.9% and 18.0% less unschedulable task sets compared to the original P-FRP scheduling. Thus the memory effect of restarted P-FRP tasks must not be ignored.

IV. CONCLUSION AND FUTURE WORK

We have proposed our preliminary research of memory-aware P-FRP model. Simulation results show the schedulability and response time improvement when considering the different execution time of a task in *cold* start and restart cases. Our ongoing research is to present more theoretical response time analysis and priority assignment research in the memory-aware P-FRP task scheduling. And since the execution time difference is likely related to data placement/locality, we will address this difference in our multi-core P-FRP task scheduling research too.

REFERENCES

[1] A. Church. An unsolvable problem of elementary number theory. American Journal of Mathematics 58, pp.345-363, 1936.

[2] C. Belwal, A. M. K. Cheng. A Utilization based Sufficient Condition for P-FRP. 9th IEEE/IFIP Int'l Conf. on Embedded and Ubiquitous Computing (EUC), 2011, pp.237-242.

[3] C. Belwal, A.M.K. Cheng. On priority assignment in P-FRP. RTAS 2010 WiP Session.

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

[5] H. C. Wong, A. Burns. Improved Priority Assignment for the Abortand-Restart (AR) Model. Technical Report YCS-2013-481, University of York, Department of Computer Science, 2013.

[6] H. C. Wong, A. Burns. Priority-based Functional Reactive Programming (P-FRP) using Deferred Abort. RTCSA 2015.

[7] https://typesafe.com/company/casestudies, 09/24/2014.

[8] K. R. Christoffersen, A. M. K. Cheng. Model-based design: Antilock brake system with priority-based functional reactive programming. RTSS 2013 WiP Session.

[9] Q. Zhou, Y. Li, X. Zou, A. M. K. Cheng, Y. Jiang. Worst Case Response Time and Schedulability Analysis for Real-Time Software Transactional Memory-Lazy Conflict Detection (STM-LCD). DPRTCPS workshop, 2015.

[10] R. Kaiabachev, W. Taha, A. Zhu. E-FRP with priorities. ACM EMSOFT 2007.

[11] S. Kleene. A theory of positive integers in formal logic. American Journal of Mathematics 57, pp.153-173 and 219-244, 1935.

[12] X. Zou, A. M. K. Cheng, Y. Jiang. A Non-Work-Conserving Model for P-FRP Fixed Priority Task Scheduling. RTSS 2015 WiP session.

[13] Y. Jiang, A. M. K. Cheng, X. Zou. Schedulability Analysis for Real-Time P-FRP Tasks Under Fixed Priority Scheduling. RTCSA 2015.

[14] Y. Jiang, Q. Zhou, X. Zou, A. M. K. Cheng, X. Zou. Minimal Schedulability Testing Interval for Real-Time Periodic Tasks with Arbitrary Release Offsets. IEEE ICESS 2014.

[15] Z. Kazemi, A. M. K. Cheng. A Scratchpad Memory-Based Execution Platform for Functional Reactive System and its Static Timing Analysis. RTAS 2015 WiP session.

[16] Z. Wan, P. Hudak. Functional reactive programming from first principles. ACM SIGPLAN PLDI 2000.