Timing Analysis of P-FRP Systems

Danielle Underwood and Albert M. K. Cheng
Real-Time Systems Laboratory
Department of Computer Science
University of Houston, Texas, USA
dlunderwood@uh.edu, cheng@cs.uh.edu

***Abstract*—P-FRP systems are relatively new, so little research has been done on how tasks are best scheduled using the P-FRP paradigm. This paper provides a background of the subjects that timing analysis must cover, and concludes with suggestions for future research to be done in this field.**

I. INTRODUCTION

Real-time systems (RTSs) are of critical importance in many areas. In fact, one of the most familiar examples of real-time systems to many is the anti-lock braking system in modern vehicles: if this system were to fail, the results could be fatal. As such, it is worth researching ways to improve RTS reaction time.

There are several ways researchers have attempted to improve the response time of these systems. These include changing the hardware the RTS uses, changing the way that information is handled, and improving the estimation of the worst-case execution (WCET) of the tasks in the system, which allows for more optimal task scheduling. In addition, there have also been multiple definitions of languages suited for real-time programming and multiple simulation tools used, each claiming some advantages and disadvantages.

All of these things should be thoroughly understood in order to make progress. The subject of FRP languages is covered in Section 2. Hardware research in RTSs is in Section 3. This is followed by information about simulation tools in Section 4. Details about attempts to more tightly bound the WCET can be found in Section 5. The related topic of scheduling is in Section 6. Section 7 describes approaches toward information handling in RTSs, and is followed by Section 8, in which ways forward in this domain are proposed. Section 9 concludes the paper.

*This work is supported in part by the US National Science Foundation (NSF) under awards No. 0720856 and No. 1219082. The first author is an NSF-REU undergraduate researcher.

II. LANGUAGE DEFINITIONS

The most widely-used functional language in real-time systems programming is Haskell. However, many believe Haskell can and should be altered to better suit it for RTSs. As such, several embedded domain-specific languages (EDSLs) have been proposed to overcome its perceived shortcomings [1, 2, 14]. P-FRP is one of these languages, designed to avoid priority inversions. The semantics of P-FRP requires that a running task be aborted when a higher-priority task arrives and the lower-priority task will later restart from the beginning as shown in Figure 1.

ID	Period	WCET
τ_1	5	2
τ_2	8	3

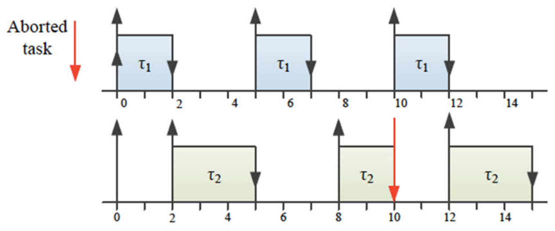

Figure 1. P-FRP abort-and-restart paradigm.

Haskell's biggest weakness in terms of RTS programming is its lack of timing guarantees. As such, EDSLs designed for these systems usually provide a limited set of instructions which have bounded WCETs. Other alterations may also be made to further customize it to RTS usage.

This is a somewhat specialized subject. Many researchers not working directly with language definitions simply sidestep the problem by using C, since Haskell does not usually compile directly to machine code, but rather first to an intermediary language. This makes it easier to analyze since most timing analyzers are designed to use C code rather than Haskell or any of its EDSLs.

III. Hardware Research

"Hardware research" here specifically refers to research in determining the superiority of one type of hardware to another. The hardware in question is usually the on-chip memory. This can be either scratchpad memory (SPM) or cache.

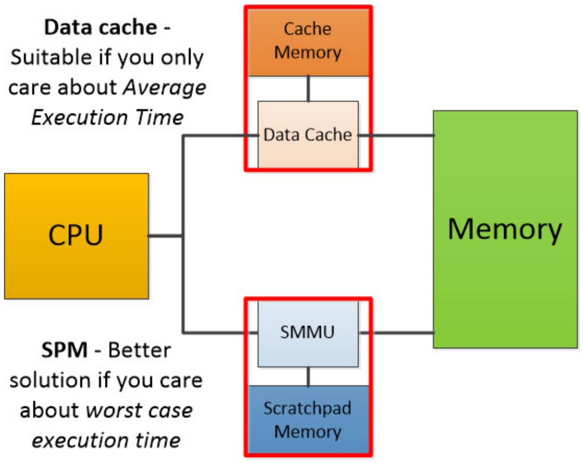
Data cache - Suitable if you only care about *Average Execution Time*
Cache Memory
Data Cache
CPU
Memory
SMMU
Scratchpad Memory
SPM - Better solution if you care about *worst case execution time*

Figure 2. Cache and scratchpad memory.

There is some debate about whether cache or SPM is superior in RTSs. [8] and [9] say that SPM is more energy-efficient and requires fewer cycles and less area. These claims are not as well-supported as they could be. Both papers model SPM using a subset of the cache, which means it automatically requires less area and energy. However, they do not justify this model very well, and only say that a SPM is similar enough to a cache without its tagging system that they believe it is a suitable substitution. While the model is probably similar enough to real SPM that the results would not be different, it would be worth repeating this research with a more exact simulation.

This belief is supported by the fact that the drastic improvements in performance found by these papers were not found in [7]. The authors of [7] found that SPM and cache were similar enough that neither had an absolute advantage over and instead found that the optimal choice depended on the tasks the system was expected to run. However, [7] is not general enough: it compares SPM with a specific type of cache using a specific loading algorithm. It is possible that under other circumstances the results would be different.

IV. Simulation Tools

Simulation tools are used for a variety of reasons. Some are designed exclusively to calculate WCET, while others calculate ACET. In addition, some tools may also project the amount of energy consumed by a system.

Chronos is one of the more widely-used tools. It simulates lower-level features in the architecture in order to calculate as tight a WCET bound as possible. It is described in [4] and [15].

Chronos was based off SimpleScalar, which is even more popular. According to the tool's website, in 2000 more than a third of papers published in related conferences used SimpleScalar [19]. Like Chronos, it models microarchitectural features for tighter bounding. The documentation for SimpleScalar can be found in [20].

Studies conducted at the University of Dortmund and the Indian Institute of Technology have used the CACTI tool made by Intel. This tool is described in [10] and [11]. The group at the University of Dortmund has also used aiT. However, this might not be the best tool for research since it is not open-source.

Unfortunately, none of these tools but aiT natively support SPM analysis. However, a group at the National University of Defense Technology created a tool called Sim-spm that extends SimpleScalar so that it will also analyze SPM [21], though this does not appear to be available online.

V. Bounding WCET

The response time of a task in a hard RTS cannot exceed a theoretical maximum without risking a missed deadline and resulting catastrophic failure. On the other hand, the theoretical maximum should be as low as possible to allow for optimal scheduling, which will allow the system the best total response time. Research in this area is usually done by creating equations to describe the memory latency of a system using ILP and then comparing the results obtained from a simulator with the expected values generated by the ILP equation.

Attempts to bound WCET in caches are complicated by the low-level features of the hardware introduced to try and reduce the ACET. Special software has been written in an attempt to make timing analysis of caches easier, but WCET estimation for caches remains difficult [4, 5, 15, 16].

One problem mentioned in Section 4 is relevant again here. No software exists that is specially made to bound WCET in SPM, despite the fact it is theoretically easier since SPM lacks the microarchitectural features that cache has. However, Chronos and aiT have both been used to model SPM and bound the WCET, and the Sim-spm extension for SimpleScalar was written expressly for this purpose.

VI. Scheduling

Scheduling improvements can be a product of research for that goal or a side effect of other studies. That is, research can be used to find the best scheduling algorithm, but in hard RTSs the best scheduling can only be achieved by having a tightly bounded WCET.

An example of scheduling improvement for its own sake can be found in [17]. Different scheduling algorithms are discussed and compared. Dynamic-priority schedulers (that is, those that can assign different priorities to different instances of the same task) are found to be superior because they only fail when any other scheduler would also fail.

Scheduling improvement as a result of something else can be seen in [18], which formalizes the definition of a priority inversion. Since scheduling relies on making sure priority inversions are rectified as soon as possible, this is important without actually being specifically about scheduling.

VII. Information Handling

Different researchers have had different ideas of how best to process instructions and data. Some have only moved either instructions [9] or data [3] onto the on-chip memory. [4] seems to suggest that the best results are obtained when both instructions and data can be copied to the on-chip memory; however, their experiment uses only cache and not SPM. Some have written to the memory dynamically [3] while others have written statically [9]. Of the two, dynamic allocation appears to be the better choice [7].

SPM specifically requires a focus on information handling. While cache loading is handled by hardware, SPM loading is handled by software and so the program needs to be annotated at points so the compiler knows what the basic blocks of the program are that can be individually loaded into the SPM.

There are several ways to generate these annotations. Some do it manually; however, this allows programmer-introduced errors to enter. To remedy this, algorithms to decide where to best annotate the program have been created.

[6], [7], [12], and [13] each have a way of deciding at what points to break up the program. [12] is an early work and just discusses one way to automate the process. The rest are designed to be superior in just one aspect. The algorithm in [6] is designed to maximize SPM usage. That of [7] is intended to minimize energy usage. The algorithm in [13] is intended for systems with both cache and SPM. Many more specialized algorithms could also be designed.

VIII. Moving Forward

Kazemi and Cheng's work [3] can be enhanced in several ways. Their work fulfills the P-FRP paradigm by dynamically allocating information to the SPM and only writing the SPM to the main memory when a commit instruction is received. This avoids the costs associated with rollback when a task is preempted [3]. However, it has no way of handling a task that uses more data than can be contained in the SPM.

One way of dealing with this is to create a type of virtual memory in the system. While virtual memory is usually compensating for a lack of RAM, in this case it would be compensating for a lack of cache or SPM. Instead of temporarily storing information on a hard drive, it would temporarily store it in the RAM. This would not improve performance over Kazemi and Cheng's work, but it would make it more general.

The execution should be simple. The compiler should allocate information to the SPM as it is needed. When the SPM is full, the LRU data should be paged out to the RAM and replaced with whatever new data is needed. Replacing the LRU data should allow for some performance improvement over putting all data accessed after the SPM is full in the off-chip memory. After a task is complete, the part of the off-chip memory used to temporarily store the data set should be marked as unused to prevent memory leakage.

This method still has some drawbacks. As mentioned, it does nothing to improve WCET; in fact, it will probably worsen performance due to both the time taken to page out information and the need to account for marking the virtual memory as unused after every task. This method only makes it possible to use Kazemi and Cheng's method in more cases. In addition, this method can still only handle so much data: if a data set is large enough to take up the SPM and more empty space than is left on the RAM by the program, it cannot be handled in this way. A solution to either problem is not readily apparent.

Another way forward is to adapt Kazemi and Cheng's work for cache. This, too, should be fairly easy to do. The cache must use a write-back policy in order to avoid data being written before a commit instruction is received. Upon receiving an abort instruction, the status bits in the cache should be cleared in order to let the memory to be used again. If a commit instruction is received, 0s should be copied to the cache in order to get it to write the data to the main memory, and then the status bits should be cleared.

This has the same problem as before: only so much can be loaded into the cache without accidental writes occurring. To avoid this, an abort should be performed if all the status bits are dirty. There is also no guarantee that the estimated WCET will decrease or be closer to the actual WCET. However, as mentioned in Section 3, there is some evidence to suggest that cache may be better than SPM in some cases.

IX. Conclusion

There are many areas in which real-time systems implemented in P-FRP can improve. Their hardware, the tightness of the WCET bound derivation, the scheduling, and the way the system handles information all have room for optimization. In addition, new execution models and analytical tools can be created to make them even better.

References

[1] Roumen Kaiabachev, Walid Taha, and Angela Yun Zhu. E-FRP with Priorities, EMSOFT'07, pp. 221-230, 2007.

[2] Chaitanya Belwal and Albert M. K. Cheng. Determining Actual Response Time in P-FRP, Proc. Thirteenth International Symposium on Practical Aspects of Declarative Languages (PADL), Austin, Texas, USA, January 24-25, 2011.

[3] Zeinab Kazemi Alamouti and Albert M. K. Cheng, ``A Scratchpad Memory-Based Execution Platform for Functional Reactive Systems and its Static Timing Analysis," 21th IEEE-CS Real-Time and Embedded Technology and Applications Symposium (RTAS) WIP, Seattle, Washington, April 14-16, 2015.

[4] Sudpita Chattopadhyay and Abhik Roychoudry. Unified Cache Modeling for WCET Analysis and Layout Optimizations, RTSS, 2013.

[5] Lars Wehmeyer and Peter Marwedel. Influence of Onchip Scratchpad Memories on WCET Prediction, 2004.

[6] Manish Verma, Stefan Steinke, and Peter Marwedel. Data Partitioning for Maximal Scratchpad Usage, ASPDAC, 2003.

[7] Stefan Steinke, Nils Grnwald, Lars Wehmeyer, Rajeshwari Banakar, M. Balakrishnan, and Peter Marwedel. Reducing Energy Consumption by Dynamic Copying of Instructions onto Onchip Memory, ISSS, 2002.

[8] Stefan Steinke, Lars Wehmeyer, Bo-Sik Lee, and Peter Marwedel. Assigning Program and Data Objects to Scratchpad for Energy Reduction, DATE, 2002.

[9] Isabelle Puaut and Christophe Pais. Scratchpad Memories Vs. Locked Caches in Hard Real-Time Systems: A Quantitative Comparison, DATE, 2007.

[10] Naveen Muralimanohar, Rajeev Balasubramonian, and Norm Jouppi. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0, IEEE/ACM Intl. Symp. On Microarchitecture, 2007.

[11] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi. CACTI 6.0: A Tool to Understand Large Caches, TR 2009.

[12] Andreas Ermedahl and Jan Gustafsson. Deriving Annotations for Tight Calculation of Execution Time, Euro-Par, 1997.

[13] Manish Verma, Lars Wehrmeyer, and Peter Marwedel. Cache-Aware Scratchpad Allocation Algorithm, DATE, 2004.

[14] Zhanyong Wan, Walid Taha, and Paul Hudak. Real-Time FRP, ICFP, 2001.

[15] Xianfeng Le, Abhik Roychoudhury, and Tulika Mitra. Modeling Out-of-Order Processors for WCET Analysis, RTSS, 2004.

[16] Friedhelm Stappert and Peter Altenbernd. Complete Worst-Case Execution Time Analysis of Straight-Line Hard Real-Time Programs, Journal of Systems Architecture, 1997 .

[17] Albert M. K. Cheng. Real-Time Systems: Scheduling, Analysis, and Verification, Wiley, 2002 and 2005.

[18] Özalp Babaoğlu, Keith Marzullo, and Fred B. Schneider. A Formalization of Priority Inversion, 1993.

[19] SimpleScalar Overview [Online]. Available: http://www.simplescalar.com/overview.html, 2004.

[20] SimpleScalar Documentation [Online]. Available: http://www.simplescalar.com/docs.html, 2004.

[21] Xiaoguang Ren, Yuhua Tang, Tao Tang, Sen Ye, Huiquan Wang, and Jing Zhou. Sim-spm: A SimpleScalar-based Simulator for multi-level SPM Memory Hierarchy Architecture, HPCC, 2010.