Cache-Aware Real-Time Scheduling Simulator: Implementation and Return of Experience

Hai Nam Tran, Frank Singhoff, Stéphane Rubini, Jalil Boukhobza
Univ. Bretagne Occidentale, UMR 6285, Lab-STICC, F-29200 Brest, France
{hai-nam.tran,singhoff,rubini,boukhobza}@univ-brest.fr

ABSTRACT

Evaluating cache related preemption delay (CRPD) in preemptive scheduling context of Real-Time Embedded System (RTES) stays an open issue despite of its practical importance. Indeed, various parameters should be taken into account such as memory layout, cache utilization, processor utilization, priority assignment and scheduling algorithm. In state-of-the-art work, dependencies amongst those parameters are not investigated with precision because of the lack of scheduling analysis tool taking them into account. In this article, we present a tool to investigate and evaluate scheduling analysis of RTES with cache memory and various scheduling parameters. The work consists in modeling guidelines and implementation of a cache-aware scheduling simulator. Implementation is made in Cheddar, an open-source scheduling analyzer, which is freely available to researchers and practitioners. Experiments are conducted in order to illustrate applicability and performance of our tool. Furthermore, we discuss about implementation issues, problems raised and lessons learned from those experiments.

1. INTRODUCTION

Cache memory is a crucial hardware component used to reduce the performance gap between processor and main memory. In the context of real-time embedded system (RTES), the popularization of processors with large size and multi-level cache motivates the proposition of verification methods handling this hardware component [14], [5], [2].

Scheduling simulation is a classical verification method performed at design step of RTES. It provides means to investigate that all timing constraints of a RTES are satisfied. To perform scheduling simulation, several assumptions are usually made in order to simplify the system model. One of them is that preemption costs are negligible. *Preemption cost* is the additional time to process interrupts, manipulate task queues and actually perform context switches.

Integrating cache memory in RTES generally enhances the whole performance in term of execution time, but unfortunately it can lead to an increase in preemption cost and execution time variability [15]. When a task is preempted, memory blocks belonging to the task could be removed from the cache. Once this task resumes, previously removed memory blocks have to be reloaded into the cache. Thus, a new preemption cost named *Cache Related Preemption Delay* (CRPD) is introduced. By definition, CRPD is the additional time to refill the cache with memory blocks evicted by the preemption. In [15], the authors showed that the cost of context switching raises from the range of $4.2\mu s$ to $8.7\mu s$ to the range of $38.6\mu s$ to $203.2\mu s$ when the size of the data sets of a program is larger than the cache. CRPD can present up to 40% of the Worst Case Execution Time (WCET) of a program [20]. Thus, taking CRPD into account is important when performing scheduling analysis of a RTES.

Problem statement: Scheduling analysis of RTES with cache memory in fixed priority preemptive scheduling context is complex because there are many parameters affecting the outcome and dependencies amongst them. Evaluating the impact of CRPD on a RTES cannot be only based on single-task analysis because it depends on correlations amongst tasks. For a set of tasks, CRPD analysis has to take into account various parameters including WCET, memory layout, cache utilization, processor utilization of tasks and priority assignment algorithm of the scheduler.

To address all parameters, many tools are involved, in two domains: WCET/Cache analysis and scheduling analysis. Each of those domains is only dedicated to a sub-part of those parameters. Unfortunately, there are no tools addressing the whole problem in the state-of-the-art work.

Contributions: We propose a tool to investigate and evaluate scheduling analysis of RTES with cache memory and various scheduling parameters. The work is implemented in Cheddar, an open-source scheduling analyzer, which is freely available to researchers and practitioners. We propose an approach to use the analysis models and results of WCET/-cache analysis tools into a scheduling analysis tool. The programming model we used is compliant with the existing one in [14], [5], [2] and [17]. Experiments are performed to illustrate performance and applicability of our tool.

The rest of the article is organized as follows. Section 2 presents background for our work. In Section 3, we give an overview of our approach. In Section 4, we present development process and detailed information about the implementation of our work. In Section 5, an evaluation in terms of applicability and performance of our proposed scheduling simulator is given. Section 6 discusses related works and Section 7 concludes the article.

EWiLi'15, October 8th, 2015, Amsterdam, The Netherlands.

2. BACKGROUND

In this section, we introduce the system model and explain the computation of preemption cost and CRPD.

We assume a uniprocessor RTES with direct-mapped instruction cache. As far as we know, instruction cache is popular in practical implementation of RTES. The assumption about direct-mapped is used to simplify the data flow analysis, it could be easily relaxed to take into account set-associativity cache as shown in [14]. There are n independent tasks $\tau_1, \tau_2, ..., \tau_n$ scheduled by a preemptive scheduler.

The additional context switch costs due to the scheduler invocation and possible pipeline-flush can be upper-bounded by a constant and included in WCET of tasks. The additional execution time due to preemption is mainly caused by cache eviction [1]. Thus, CRPD can be used to refer to the preemption cost. CPRD is bounded by $g \cdot BRT$, where g is an upper bound on the number of cache block reloaded due to preemption, and BRT is an upper-bound on the time necessary to reload a memory block in the cache (block reload time). To analyze the effect of preemption on a preempted task, Lee et al.[14] introduced the concept of useful cache block (UCB):

Definition 1. A memory block m is called a useful cache block (UCB) at program point P, if m may be cached at P and m may be reused at program point P' after P that may be reached from P without eviction of m on this path.

The number of UCB at program point P gives an upper bound on the number of additional reloads due to a preemption at P. The maximum possible preemption cost for a task is determined by the program point with the highest number of UCB. In [23], the authors exploit the fact that for the i-th preemption, only the i-th highest number of UCB has to be considered. However, as shown in [1] and [3], a significant reduction typically only occurs at a high number of preemptions. Thus, we only consider the program point with highest number of UCB.

The impact of preempting task is given by the number of cache blocks that the task may evict during its execution. Busquet et al.[5] introduced the concept of evicting cache block (ECB):

Definition 2. A memory block of the preempting task is called an evicting cache block (ECB), if it is accessed during the execution of the preempting task.

The notation $\mathrm{UCB_i}$ and $\mathrm{ECB_i}$ are used to present the set of UCB and ECB of a task τ_i. Assume that the sets of UCB and ECB of each task are preliminarily computed, $\mathrm{UCB'_i}$ is the set of UCBs currently in the cache of the preempted task. $\gamma_{i,j}$ is the preemption cost (i.e. the CRPD) when a task τ_j directly preempts task τ_i. In case of a preemption between two tasks τ_i and τ_j, $\gamma_{i,j}$ is computed by:

$$\gamma_{i,j} = \mathrm{BRT} \cdot |\ \mathrm{UCB'_i} \cap \mathrm{ECB_j}\ | \tag{1}$$

However, in case of nested preemptions, a task τ_j can preempt more than one task. Thus, computation of CRPD must take all preempted tasks into account.

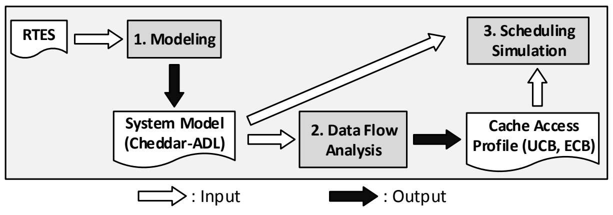

Figure 1: Approach overview

3. OUR APPROACH

Our approach consists of three steps as shown in Fig. 1. First, we model a RTES with components required to apply analysis methods stated in Section 2. Those include hardware and software components. Hardware components are processor, core and cache memory. Software components are tasks and control flow graphs of tasks.

Second, from this model, we apply data flow analysis presented in [14] in order to compute the set of UCB and ECB of each task, which is called cache access profile in the sequel. A detailed description of these first and second steps could be found in [25], which is a preliminary work toward cache integration.

Third, system model with computed cache access profiles are loaded into the scheduling simulator. Scheduling simulation is done and provides various data such as feasibility of the system, worst case response time of tasks, number of preemption, CRPD per task and total CRPD.

In our work, the cache utilization of a program at step 1 is modeled at both low level, which is the control flow graph of program produced by a WCET analysis tool, and at high level, which is a pre-computed set of UCB and ECB. We expect the user to re-use results produced by a WCET analysis tool within Cheddar. We can extract all information related to CRPD from the scheduling simulator as written above to fully investigate the impact of CRPD. Detailed implementation of each step is presented in the next section.

4. IMPLEMENTATION

In this section, we present the implementation of our approach. We introduce our framework, discuss the development process and point out several implementation issues.

The work is proceeded in the context of the Cheddar project [22]. Cheddar is an open source real-time scheduling analysis tool. Classical feasibility tests, scheduling algorithms and a scheduling simulator for real-time systems are implemented in Cheddar. System architectures are defined with Cheddar Architecture Description Language (Cheddar-ADL). The Cheddar-ADL meta-model is specified with the modeling language EXPRESS. Cheddar class files are automatically generated by the tool Platypus [21] through a model-driven process. The Cheddar meta-model defines hardware components such as: *processor*, *core* and *shared_resource*; and software components such as: *task* and *task_group* [11].

The development process consists of three steps. First, Cheddar-ADL is extended to model RTES with cache memory. Second, we implement data flow analysis presented in [14] to compute cache access profiles of tasks. Third, a cache-aware scheduling simulator is implemented by extending the scheduling simulator of Cheddar. In [25], we presented in detail the extension of Cheddar-ADL and the implementation of data flow analysis method. In this article, we only provide a summary of our models and focus on the implementation of the cache-aware scheduling simulator and its application.

4.1 Cache and Cache Access Profile Model

To support cache aware scheduling analysis, Cheddar-ADL has been extended with the entities below:

```
ENTITY Generic_Cache
    cache_size : Natural;
    line_size : Natural;
    associativity : Natural;
    block_reload_time : Natural;
    cache_category : Cache_Type;
ENTITY Generic_Task
    task_type : Natural;
    capacity : Natural;
    deadline : Natural;
    priority : Priority_Range;
    cache_access_profile_name : String;
    cfg_name : String;
ENTITY Cache_Access_Profile
    UCBs : Cache_Blocks_Table;
    ECBs : Cache_Blocks_Table;
ENTITY CFG
    nodes : CFG_Nodes_Table;
    edges : CFG_Edges_Table;
ENTITY Basic_Block SUBTYPE OF (CFG_Node);
    instruction_offset : Natural;
    instruction_capacity : Natural;
```

Cache memory is modeled by classical attributes such as cache size, line size, associativity and block reload time. From these models of cache memory and control flow graphs, we can compute the memory layout, cache utilization and cache access profile of tasks.

A task is modeled by an entity with classical attributes such as *capacity*, *deadline*, *priority* and $offsets$. The *task_type* specifies the type of a task, such as *periodic*, *aperiodic*, *sporadic* or *poisson*. A detailed information of hardware and software models of Cheddar could be found at [11].

Two attributes, *UCBs* and *ECBs* are added to model cache access profile. They are the set of UCBs and ECBs of a task, respectively. The model is designed according to two assumptions. (1) First, a task is assumed to have a fixed set of UCB and ECB. UCB is computed by the program point with highest number of UCB. We acknowledge that it is only applied to single path program with a large percentages of computational requirement in a loop. (2) Second, any partial execution of a task uses all its ECBs and UCBs.

In order to compute the set of UCB and ECB of a task, we need its control flow graph (which is modeled by the CFG EXPRESS entity). Each node in the CFG, which is called a basic block, stores information of the capacity and the location in main memory of its assembly instruction. At the moment, our analysis method takes into account direct mapped instruction cache because instruction cache access pattern is simpler to be computed from the CFG of a program.

4.2 Cache Access Profile Computation

The input of the algorithm is an annotated CFG of a program. Going from the input, our tool analyzes and computes the set of UCB and ECB of this program.

ECBs are computed by taking into account memory blocks accessed in the execution of a program. In our models, position and size of assembly instructions of each basic blocks are stored. Knowing these data and associativity of the cache, we can compute easily the set of ECBs.

UCBs are computed by applying data flow analysis presented in the work of [14]. In our work, the set of UCB of each basic blocks are computed. The set of UCB of the basic blocks with highest number of UCB are chosen to represent the set of UCB of a program.

4.3 Cache-Aware Scheduling Simulator

The scheduling simulator in Cheddar works as follows. First, a system architecture model, including hardware/software components, is loaded. Then, the scheduling is computed by three successive steps: computing priority, inserting ready task into queues and electing task. The elected task will receive the processor for the next unit of time.

The scheduling simulator records different events raised during the simulation, such as task releases, task completions and shared resources lockings or unlockings. The result of the scheduling analysis is the set of events produced at simulation time.

The scheduling simulator of Cheddar is extended as follows. First, we extend the set of events Cheddar can produce. For example, an event PREEMPTION, which is raised when a preemption occurs, is added. Second, event RUNNING_TASK, which is raised when a task executes, is extended with the assumption about CRPD that any partial execution of a task uses all its ECBs and UCBs.

The pseudo code of the event handler is written below. The notation τ_i.cUCB represents the set of UCBs of task τ_i in the cache. It is computed from a system model at scheduling simulation time. The function Remove() at line 8 is used to remove an element from a set.

```
1  event SCHED_START
2      for each task $\tau_i$ loop
3          $\tau_i$.cUCB $\leftarrow$ $\tau_i.UCB$
4      end loop
5  event PREEMPTION
6      $\tau_j \leftarrow$ preempting_task
7      for each task $\tau_i$ preempted loop
8          $\tau_i$.cUCB $\leftarrow$ Remove($\tau_i$.cUCB, $\tau_j$.ECB)
9      end loop
10 event RUNNING_TASK
11     $\tau_i \leftarrow$ executing_task
12     CRPD $\leftarrow$ ($\tau_i.UCB$ $-$ $\tau_i$.cUCB) $*$ Miss_Time
13     $\tau_i$.cUCB $\leftarrow$ $\tau_i.UCB$
```

The event handler is described as follows. At the start of the scheduling simulation, a SCHED_START event is raised. WCET of a task is assumed to include the cache block reloading time when the task is executed non-preemptively. So, on event SCHED_START, the set of UCBs of a task is assumed to be filled.

When a preemption occurs, a PREEMPTION event is raised and the simulator computes the evicted UCBs of preempted tasks by taking into account the ECBs of preempting task. The scheduler keeps track of the number of UCBs in the cache of each task. At this event, the CRPD is not computed yet.

When a task executes, a RUNNING_TASK event is raised. The scheduler first checks if all the UCBs of this task are loaded into the cache. If so, the task continues its execution. If not, the task reloads the evicted UCBs. The CRPD is added to the remaining capacity of the task itself. In our implementation, CRPD is not added to the capacity of preempted tasks at the preemption point but at the instant, of which those tasks resume execution.

4.4 Implementation Issues

Several issues were raised when designing and implementing the simulator. Most of them are related to **mixing timing specifications of different orders of magnitude**. Others are related to **tools interoperability**.

Mixing timing specifications of different orders of magnitude makes the computation of the *feasibility interval* complex. We recall that a feasibility interval is an interval of time for which testing of task feasibility is needed [10]. In practice, cache block reload time is significantly smaller than period or capacity of a task. In Cheddar, we do not prescribe 1 unit of time is equivalent to $1ms$ or $1\mu s$, which are the granularity of task period and block reload time. The scheduling simulation interval needed to verify the schedulability of a task set could be significantly large if a μs is chosen as a time unit. A solution in practice is to design systems with harmonic task sets in order to minimize the feasibility interval; however, it is clearly not always possible. In addition, instead of using $1\mu s$, we use the cache block reload time as a base value for 1 unit of time.

A large scheduling simulation interval also raises issues regarding performance and scalability. Even with harmonic task sets, the tool must be able to perform scheduling simulations in a large interval to overcome the difference between cache block reload time and task period, which may be CPU and memory expensive. As Cheddar stores scheduling simulation results into XML files, it can also be I/O intensive. To reduce memory and I/O overhead, we selected a subset of events the simulator has to handle and store.

A second issue we were facing is about tool interoperability. The input data of the CRPD analysis in our tool is designed to be compatible with data provided by a WCET analysis tool. We also support data input in XML format, but, at the moment, we do not enforce tool interoperability and we expect to investigate WCET tools in order to overcome this issue.

5. EXPERIMENT AND DISCUSSION

In this section, we perform experiments to demonstrate that our tool can handle parameters compliant with the existing works in [14], [5], [2]. In addition, we discuss about the dependency between CRPD and scheduling parameters. Furthermore, we point out that our tool can run CRPD optimization techniques by taking an example of memory layout optimization by simulated annealing following the work of [17]. We also provide performance and scalability tests of the tool.

Experiments are performed with randomly generated task sets. The base configuration of our experiments is as follows. Task period and cache utilization generation of our experiments is based on the existing work in [1]. Task periods are uniformly generated from 5ms to 500ms, as found in most automotive and aerospace hard real-time applications [1]. Generated task sets are harmonic in order to have a low feasibility interval and scheduling simulation period. Task deadlines are implicit, i.e. $\forall i : D_i = T_i$. Processor utilization values (PU) are generated using the UUniFast algorithm [4]. Task execution times are set based on the processor utilizations and the generated periods: $\forall i : C_i = U_i \cdot T_i$, where U_i is the processor utilization of task i. Task offsets are uniformly distributed from 1 to 30 ms.

Cache memory is direct mapped. Number of cache blocks is equal to 256 and block reload time is 8μs. Cache usage of each task is determined by the number of ECBs. They are generated using UUniFast algorithm for a total cache utilization (CU) of 5. UUniFast may produce values larger than 1 which means a task fills the whole cache. ECBs of each tasks are consecutively arranged from a cache set. For each task, the UCBs are generated according to a uniform distribution ranging from 0 to the number of ECB multiplied by a reuse factor (RF). If the set of ECB generated exceeds the number of cache sets, the set of ECB is limited to the number of cache sets. For the generation of the UCBs, the original set of ECB is used.

5.1 CRPD Analysis with Priority Assignment and Processor Utilization

In this experiment, we present CRPD analysis with different priority assignments, scheduling algorithms and processor utilization (PU). In addition, we discuss about the impact of changing priority assignment/scheduling algorithm and increasing PU to CRPD.

The configuration of this experiment is as follows. PU is varied from 0.5 to 0.95 in steps of 0.05. RF is fixed at 0.3. For each value of PU, we perform scheduling simulations with 100 task set and compute the average number of preemptions and average total CRPD in a scheduling interval of 1000ms. Experiments are conducted with two priority assignment algorithms: Rate Monotonic (RM) and another we called PA*, which assigns the highest priority level to the task with the largest set of UCB. In addition, we take into account Earliest Deadline First (EDF) scheduler.

The result of this experiment is sketched in Fig. 2. As the graph illustrates, the number of preemptions and the preemption cost increases steadily from the processor utilization of 50% to 80%. After this point, there is a downward trend in the preemption cost and in the number of preemptions of EDF while there is an upward trend in those data for RM and PA*. Observed from the scheduler, when PU is larger than 80%, many task sets are not schedulable.

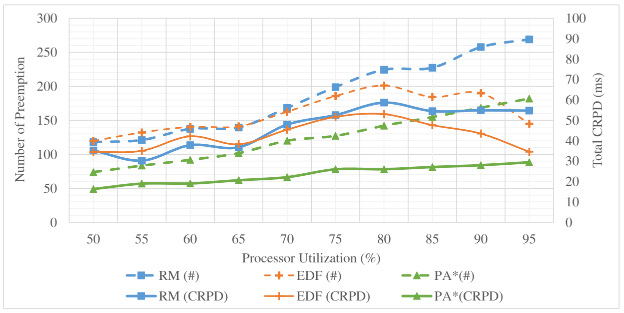

Figure 2: Varying PU, RF=0.3

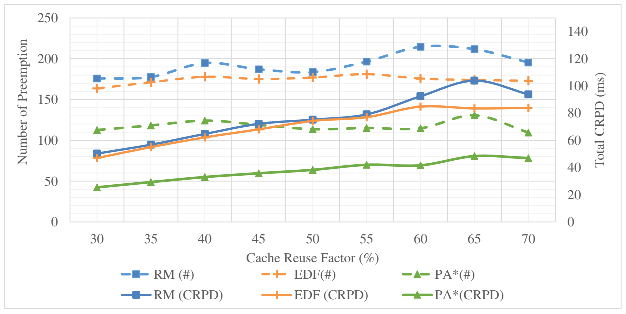

Figure 3: Varying RF, PU=0.7

In conclusion, first, when PU increases, the total number of preemption and CRPD also increase. However, the change is not linear. Second, a priority assignment algorithm with less number of preemption tends to give lower total CRPD. EDF and PA* generate less preemptions and CRPD than RM. In fact, to enforce the fixed priority order, the number of preemptions that typically occurs under RM is higher than that under EDF [6]. From this experiment, we see that CRPD depends on the chosen priority assignment or scheduler.

In addition, this experiment shows that both scheduling analysis and CRPD analysis should be performed jointly. PA*, a priority assignment taking CRPD into account has a significant lower total CRPD. The decrease in total CRPD of PA* with RM and EDF is roughly 30ms on a scheduling interval of 1000ms. However, comparing to RM and EDF, feasibility constraints of tasks are not respected with PA*, only total CRPD is reduced. In other words, the number of task that missed deadline in PA* is higher than that of RM and EDF. In [26], we proposed a priority assignment heuristic to take into account both feasibility constraints and CRPD.

5.2 CRPD Analysis with Priority Assignment and Cache Reuse Factor

In this experiment, we observe the change in the result of CRPD analysis when varying RF parameter instead of PU parameter. The configuration of this experiment is similar to the first experiment, except that PU is fixed at 0.7 and RF is varied from 0.3 to 0.7. For each value of RF, we perform scheduling simulations with 100 task set and compute the average number of preemptions and average total CRPD in a scheduling interval of 1000ms.

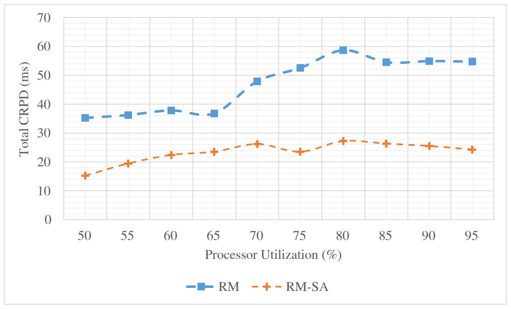

Figure 4: Total CRPD with and without memory layout optimization

The result of this experiment is shown in Fig. 3. We get a similar observation with the first experiment in terms of number of preemption and total CRPD regarding those three priority assignment algorithms. However, when varying RF, the change in number of preemption is less significant, with a maximum difference of 50 preemptions; and the change in total CRPD is more significant, with a maximum difference of 50 ms, than when varying PU (with maximum difference of number of preemption and total CRPD are 150 preemptions and 35 ms, respectively).

To conclude, experiment 5.1 and 5.2 showed that our tool can perform scheduling simulation of RTES with cache with various scheduling parameters and can be used to study the dependencies amongst those parameters.

5.3 CRPD Analysis with Memory Layout Optimization by Simulated Annealing

The objective of this experiment is to show that users can use CRPD optimization approaches with our tool. We apply memory layout optimization by simulated annealing (SA) based on the work of [17] with our generated task sets. In our experiment, the objective of SA is to lower the total CRPD after a scheduling simulation over a scheduling interval of 1000ms.

For each iteration of SA, we perform a swap in memory layout between two random tasks. Changes are made to the layout of tasks in memory, and then mapped to their cache layout for evaluation. The total CRPD is computed by scheduling simulation. The optimum layout is the layout which has the lowest total CRPD. Initial temperature of SA is 1.0, and after every iteration, it is reduced by multiplying it by a cooling rate of 0.5 until it reaches the target temperature of 0.2. The number of iteration for each temperature is 10.

The result of this experiment is shown in Fig. 4. From the graph, we can see the impact of memory layout optimization to total CRPD. We can reduce roughly 30-50% of total CRPD. To sum up, this experiment shows that our tool allows users to perform a specific optimization of CRPD for a given scheduling algorithm.

5.4 Performance/Scalability Analysis

The objective of this experiment is to test the performance and the scalability of our tool when scheduling simulation

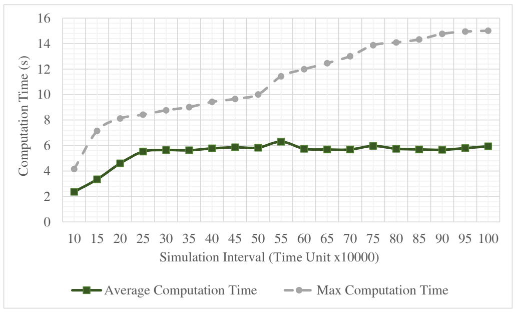

Figure 5: Computation time of the simulator

interval increases. In general, there are four factors affecting the performance of a scheduling simulator: (1) the number of tasks, (2) the scheduling simulation interval, (3) the cache size and (4) the number of events. The first three factors depend on the chosen RTES. The number of events depends on characteristics of the RTES; for example, a higher processor utilization means a higher number of preemption events. In this experiment, we choose to test a system model of 10 tasks and 256 cache blocks. Processor utilization is set to 70 %. Scheduling simulation is ranging from 100,000 to 1,000,000 units of time where 1 unit = 8μs.

Fig. 5 displays results of our experiment on a PC with Intel Core i5-3360 CPU, 4 GBs of memory, running Ubuntu 12.04. For each simulation interval, 100 task sets are generated. We perform scheduling simulation and compute the maximum and average computation time.

As we can see, while maximum computation time increases slightly when simulation interval increases, average computation time only fluctuates around 6 seconds. This shows that the tool is scalable when simulation interval is high.

6. RELATED WORKS

In this section, we present several real-time scheduling analysis and WCET analysis tools.

MAST[12] is a modeling and analysis suite for real-time applications. The hardware component abstraction of MAST model is generic and it includes processing resources and shared resources. The shared resource component is not supposed to model a cache memory unit. However, MAST considers the overhead parameters of the components that may be used to model CRPD.

STORM[27] and YARTISS [7] are scheduling simulation tools mainly designed for evaluating and comparing real-time scheduling algorithm for multiprocessor architectures. SymTA/S[13] and RealTime-at-Work [1] are model-based scheduling analysis tools targeting automotive industry. The hardware components supported in those tools are specific to their domains (ECU, CAN and AFDX Networks). To the best of our knowledge, the support for cache memory does not exists in the tools above.

SimSo[8] is a scheduling simulation tool. It supports cache sharing on multi-processor systems. It takes into account impact of the caches through statistical models and also the direct overheads such as context switches and scheduling decisions. The memory behavior of a program is modeled based on Stack Distance Profiles (SDP) - the distribution of the stack distances for all the memory accesses of a task, where a stack distance is by definition the number of unique cache lines accessed between two consecutive accesses to a same line [18]. The difference between SDP and our model is that SDP is achieved by on-line monitored counters such as valgrind [19] while UCB and ECB are achieved by an offline WCET analysis tools as below. At the moment, there is no archived comparison between the two. UCB analysis with scheduling simulation could be more pessimistic but safer because it takes into account the program point with largest number of UCB.

Several WCET analysis tools allow designers to perform cache analysis. SymTA/P[24], HEPTANE[9], Chronos[16] and aiT[2] are examples of them. UCB computation of a program is supported by aiT. The analysis of those tools are based on program path analysis of the control flow graph of the program. It is compliant with the requirement of our proposed tool.

In conclusion, WCET tools focus on the evaluation of program's control flow graph to compute the WCET and also a few tools can compute cache access profile. The analysis result could be used as an input for a scheduling analysis tool. In the domain of real-time scheduling analysis, the support for cache and CRPD when performing scheduling analysis is not very well specified. As far as we know, only SimSo clearly supports scheduling simulation with cache analysis based on SDP. Then, we proposed a tool available to the community which can either compute cache access profile of a task from its control flow graph or re-use information obtained from a WCET/cache analysis tool to perform scheduling analysis. Our model is compliant with existing work in [14], [5], [2] and [17]. In addition, because Cheddar provides a large set of scheduling analysis methods, we can fully investigate the dependency between CRPD and other scheduling parameters in order to either adjust or optimize a RTES design to meet its timing constraints.

7. CONCLUSIONS

In this article, we presented an approach to implement a cache-aware scheduling simulator. The work was proceeded in the context of the Cheddar real-time scheduling analyzer, which is open-source, freely available to researchers and practitioners that want to investigate scheduling analysis of RTES with cache memory. Our solution consists of three parts: modeling of the cache memory and cache access profile, implementing cache analysis methods and performing scheduling simulation. We extended Cheddar to be able to deal with cache memory and illustrated the dependency between cache and other scheduling parameters. The source code of the presented work is available under GNU GPL licence at `http://beru.univ-brest.fr/svn/CHEDDAR/trunk/src/`.

There are open problems we are aiming to address in the future. The first one concerns the refinement of other CRPD analysis methods. Several improvements have been pro-

[1]RealTime-at-Work, http://www.realtimeatwork.com/

[2]AbsInt Inc., http://www.absint.com/

posed in [2] to reduce the upper-bound of the CRPD. In addition, we plan to compare our approach with the approach based on Stack Distance Profile in [8]. Second, we are going to study the problem of feasibility interval in the context of RTES with cache Nowadays, the feasibility interval and how long should we run the scheduling simulation are still open questions. Our aim is to investigate this problem with the support of the cache-aware scheduling simulator.

8. REFERENCES

[1] S. Altmeyer, R. I. Davis, and C. Maiza. Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. *Real-Time Systems*, 48(5):499–526, 2012.

[2] S. Altmeyer and C. Maiza Burguière. Cache-related preemption delay via useful cache blocks: Survey and redefinition. *Journal of Systems Architecture*, 57(7):707–719, 2011.

[3] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, and G. Buttazzo. Optimal selection of preemption points to minimize preemption overhead. In *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*, pages 217–227. IEEE, 2011.

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

[5] J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, and A. Wellings. Adding instruction cache effect to schedulability analysis of preemptive real-time systems. In *Proceedings of the* 2^{nd} *IEEE Real-Time Technology and Applications Symposium (RTAS)*, pages 204–212, 1996.

[6] G. C. Buttazzo. Rate monotonic vs. edf: judgment day. *Real-Time Systems*, 29(1):5–26, 2005.

[7] Y. Chandarli, F. Fauberteau, D. Masson, S. Midonnet, M. Qamhieh, et al. Yartiss: A tool to visualize, test, compare and evaluate real-time scheduling algorithms. In *Proceedings of the 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, pages 21–26, 2012.

[8] M. Chéramy, A.-M. Déplanche, P.-E. Hladik, et al. Simulation of real-time multiprocessor scheduling with overheads. In *International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*, 2013.

[9] A. Colin and I. Puaut. Worst-case timing analysis of the rtems real-time operating system. *Rapport No. PI1277, IRISA, France*, 1999.

[10] L. Cucu and J. Goossens. Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors. In *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 397–404, 2006.

[11] C. Fotsing, F. Singhoff, A. Plantec, V. Gaudel, S. Rubini, S. Li, H. N. Tran, L. Lemarchand, P. Dissaux, and J. Legrand. *Cheddar Architecture Description Language*, 2014.

[12] M. González Harbour, J. Gutiérrez García, J. Palencia Gutiérrez, and J. Drake Moyano. Mast: Modeling and analysis suite for real time applications. In *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pages 125–134. IEEE, 2001.

[13] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis–the symta/s approach. *IEE Proceedings-Computers and Digital Techniques*, 152(2):148–166, 2005.

[14] C.-G. Lee, H. 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. *IEEE Transactions on Computers*, 47(6):700–713, 1998.

[15] C. Li, C. Ding, and K. Shen. Quantifying the cost of context switch. In *Proceedings of the 2007 workshop on Experimental computer science*. ACM, 2007.

[16] X. Li, Y. Liang, T. Mitra, and A. Roychoudhury. Chronos: A timing analyzer for embedded software. *Science of Computer Programming*, 69(1):56–67, 2007.

[17] W. Lunniss, S. Altmeyer, and R. I. Davis. Optimising task layout to increase schedulability via reduced cache related pre-emption delays. In *Proceedings of the 20th International Conference on Real-Time and Network Systems*, pages 161–170. ACM, 2012.

[18] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems journal*, 9(2):78–117, 1970.

[19] N. Nethercote and J. Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. In *ACM Sigplan notices*, volume 42, pages 89–100, 2007.

[20] R. Pellizzoni and M. Caccamo. Toward the predictable integration of real-time cots based systems. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 73–82. IEEE, 2007.

[21] A. Plantec and F. Singhoff. Refactoring of an ada 95 library with a meta case tool. In *ACM SIGAda Ada Letters*, volume 26, pages 61–70. ACM, 2006.

[22] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: a flexible real time scheduling framework. *ACM SIGAda Ada Letters*, 24(4):1–8, 2004.

[23] J. Staschulat, S. Schliecker, and R. Ernst. Scheduling analysis of real-time systems with precise modeling of cache related preemption delay. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 41–48, 2005.

[24] J. Staschulat, S. Schliecker, and R. Ernst. Scheduling analysis of real-time systems with precise modeling of cache related preemption delay. In *Euromicro Conference on Real-Time Systems (ECRTS)*, Palma de Mallorca, Spain, 2005.

[25] H. N. Tran, F. Singhoff, S. Rubini, and J. Boukhobza. Instruction cache in hard real-time systems: modeling and integration in scheduling analysis tools with AADL. In *Proceedings of the* 12^{th} *IEEE International Conference on Embedded and Ubiquitous Computing*, Milan, Italy, 2014.

[26] H. N. Tran, F. Singhoff, S. Rubini, and J. Boukhobza. Addressing cache related preemption delay in fixed priority assignment. In *Proceedings of the 20th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Luxembourg, 2015.

[27] R. Urunuela, A. Deplanche, and Y. Trinquet. Storm, a simulation tool for real-time multiprocessor scheduling evaluation. In *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, 2010.