Optimizing Temporal Decoupling using Event Relevance

Lukas Jünger
juenger@ice.rwth-aachen.de
RWTH Aachen University

Carmine Bianco
carmine.bianco@synopsys.com
Synopsys GmbH

Kristof Niederholtmeyer
kristof.niederholtmeyer@synopsys.com
Synopsys GmbH

Dietmar Petras
dietmar.petras@synopsys.com
Synopsys GmbH

Rainer Leupers
leupers@ice.rwth-aachen.de
RWTH Aachen University

PREPRINT - published in ASPDAC 2021, doi: 10.1145/3394885.3431419

ABSTRACT

Over the last decades, HW/SW systems have grown ever more complex. System simulators, so called virtual platforms, have been an important tool for developing and testing these systems. However, the rise in overall complexity has also impacted the simulators. Complex platforms require fast simulation components and a sophisticated simulation infrastructure to meet today's performance demands. With the introduction of SystemC TLM2.0, temporal decoupling has become a staple in the arsenal of simulation acceleration techniques. Temporal decoupling yields a significant simulation performance increase at the cost of diminished accuracy. The two prevalent approaches are called static quantum and dynamic quantum. In this work both are analyzed using a state-of-the-art, industrial virtual platform as a case study. While dynamic quantum offers an ideal trade-off between simulation performance and accuracy in a single-core scenario, performance reductions can be observed in multi-core platforms. To address this, a novel performance optimization is proposed, achieving a 14.32% performance gain in our case study while keeping near-perfect accuracy.

1 INTRODUCTION

The overall rise in HW/SW system complexity and strong demands regarding safety and security have reinforced the need for thorough system validation in many industries. For example, in the automotive sector rigorous requirements, laid down in ISO 26262 [1], lead to a strong demand for high-reliability compute platforms. To ease the development of these intricate HW/SW systems, software simulators, commonly referred to as Virtual Platforms (VPs), are deployed. VPs bear several advantages compared to traditional hardware prototypes. They allow for deep introspection and offer a high flexibility regarding hardware and software changes. When VPs are made available early in the design cycle, software development can start months before the physical prototype is available and hardware changes can still be performed, enabling HW/SW codesign. In addition, VPs are easily scalable once their construction is complete, while usually only a limited quantity of hardware prototypes is available. This enables early large-scale system testing.

However, the growing complexity of the simulated HW/SW system is also reflected in the VP. Unfortunately, this often impacts simulation performance, diminishing VP usability. Numerous ways of improving performance have been proposed, which usually come at the price of reduced accuracy. This issue also affects the *temporal decoupling* technique introduced with SystemC TLM2.0 and its loosely-timed coding style [2]. Here, simulation components are allowed to run ahead of the global simulation time instead of keeping tight synchronization. The maximum amount of simulation time a component is allowed to run ahead is referred to as the *quantum*. Temporally decoupled components synchronize their local time with the global simulation time at the end of each quantum. When simulating processors the quantum may also be given in cycles or instructions. To increase performance, the quantum should be as large as possible to reduce context switching. However, a large quantum reduces simulation accuracy, as events may be handled too late. Therefore, deploying temporal decoupling is not trivial. This work makes the following contributions to addressing this issue:

(1) Thorough analysis of the two prevalent SystemC TLM2.0 temporal decoupling schemes
(2) Novel performance optimization in temporally decoupled simulations while keeping near-perfect accuracy
(3) Representative case study to show the potential of our method using an industrial, state-of-the-art VP

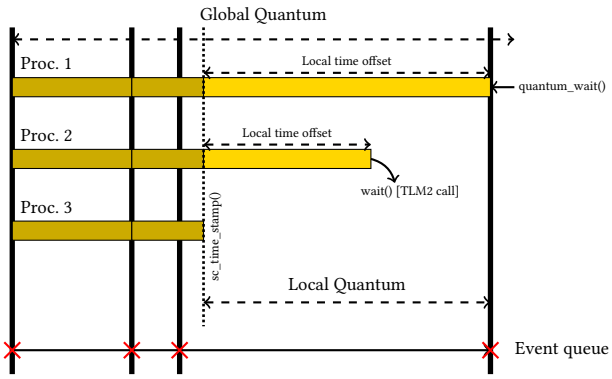

Figure 1: Dynamic quantum execution

2 RELATED WORK

Temporal decoupling increases performance by reducing context switches. However, communication between components is impacted. At large temporal offsets event notifications between component might be handled too late or missed entirely. Therefore, the fundamental question is how to aptly select the temporal offset or when to synchronize time between components. Besides the static quantum [2] and the dynamic quantum described in Section 3, several works seek to address these issues.

Gläser et al. propose a predictive approach in [5]. In their work a single-core scenario is assumed with a source generating events at

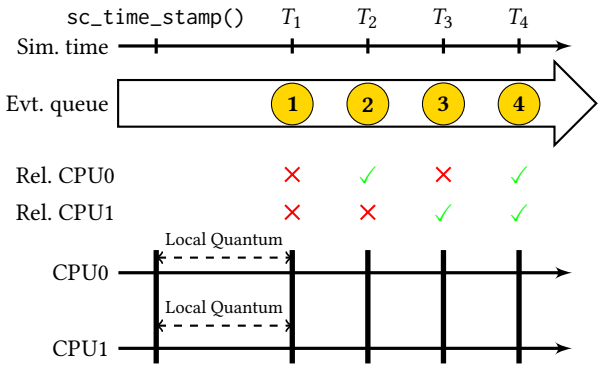

Figure 2: Dynamic quantum boundaries

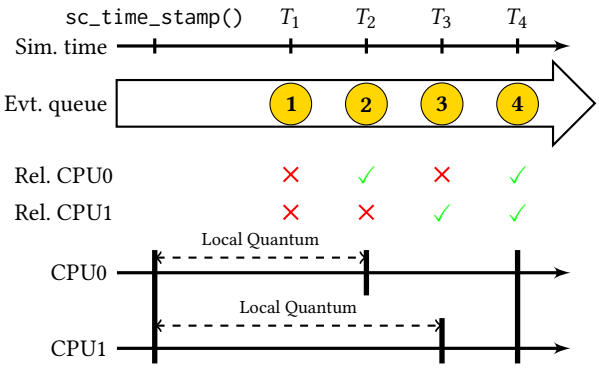

Figure 3: Quantum boundaries with ERO

random points in simulation time. To optimize the quantum size and set the next synchronization boundary, a predictor is used. A safety margin is subtracted from the predicted event occurrence time and the next boundary is set there. At this point the processor model switches to cycle-accurate simulation until the event is triggered. Here, the accuracy-performance trade-off is controlled by the safety margin size. A larger margin leads to more time spent in slower cycle-accurate simulation, but less missed events.

Another approach that strives to avoid the accuracy issue altogether is proposed by Jung et al. [7]. Traditional TLM2.0 temporal decoupling is combined with a rollback mechanism. The quantum is executed speculatively, if an event would be missed the quantum size is reduced and the rollback mechanism is deployed to execute it again. Thus, temporal errors can be avoided completely. However, this technique has a significant performance impact.

In parallel, multi-threaded SystemC simulation, the problem of synchronization arises as well. Weinstock et al. propose to partition the simulation in several clusters that are executed in parallel [11]. Each cluster adopts a static quantum approach, while events notified across cluster boundaries carry a notification delay, which is a system parameter. By ensuring that the static quantum value is smaller than this notification delay, high accuracy can be achieved within single clusters. Performance can be increased using large delays, but this diminishes the overall accuracy. Manual platform partitioning and selection of the notification delay is required. Similar approaches are taken in [8, 12]. Another way to avoid the issue in parallel simulations is proposed by Schmidt et al. [10]. In their work threads are partitioned in segments that are bounded by calls to SystemC's `wait()` function. These segments are analyzed for data dependencies. If no data dependencies are detected between segments, they can be computed in parallel. Temporal errors are avoided, but the parallel segments are smaller and higher runtime overhead for context switches is incurred. Schuhmacher et al. propose a similar approach with *parSC* [9]. Here, parallel simulation is confined within each delta cycle. For SystemC simulations distributed over networked compute nodes, Combes et al. propose a similar parallel simulation approach [4].

3 MOTIVATION

The *static quantum* is the standard temporal decoupling technique of SystemC [2]. Using this technique, synchronization points are set at integer multiples of a global quantum parameter, shared by all SystemC processes. It may be changed during simulation, but usually remains more or less constant. The local quantum is the offset between current simulation time and the next synchronization point. This approach carries several disadvantages. First and foremost, the optimal quantum size has to be determined manually. In addition, the optimal quantum size may change over the duration of the simulation. Transaction-heavy phases may require a tighter synchronization than computation phases. Furthermore, the standard does not specify whether a synchronization should occur after every inter-component transaction. Doing so preserves accuracy, but has an impact on performance due to more context switching. The intermediate synchronization of processes that have not exhausted their quantum is referred to as "breaking the quantum".

To overcome the disadvantages of the static quantum approach, the so called *dynamic quantum* was introduced. It differs from the static quantum mainly by its choice of synchronization points. Instead of placing the quantum boundaries at integer multiples of the global quantum, they are placed at the time of the next scheduled SystemC timed event notification. A global quantum value still exists as an upper limit for the quantum size. Compliance with the SystemC standard is preserved, which makes explicit provisions for overriding the local quantum computation.

An example is depicted in Figure 1. Three SystemC processes are executing. The current simulation time is marked by the dotted line labeled with `sc_time_stamp()`. The first process has executed until its local time has reached the time of the next pending event notification. It has called the special `quantum_wait()` function to signal that it has ended its quantum and is now waiting for synchronization. The local time of the second process has not yet reached the time of the next pending event notification, but the process has broken the quantum by calling the `wait()` function, e.g. after finishing a transaction, and is now waiting for the rest of the simulation to synchronize. When the third process starts executing it will first catch up until it has synchronized with process 2. Afterwards, both processes can execute until either the time of the next scheduled event is reached or the quantum is broken again.

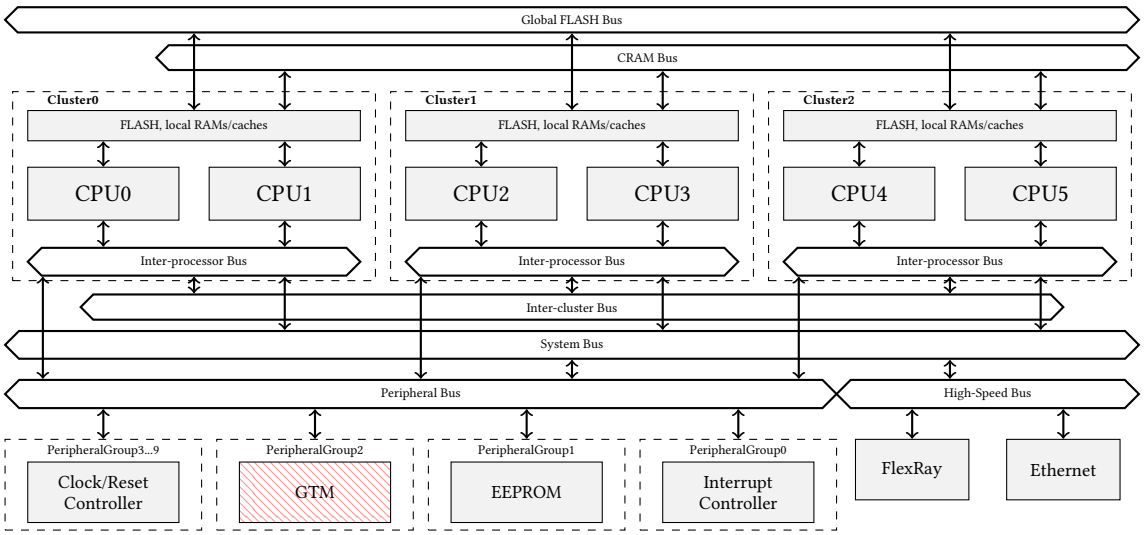

Figure 4: Overview of Renesas RH850-based VP used in the case study. The Bosch GTM highlighted is highlighted in red.

The primary advantage of the dynamic quantum approach is that it yields a near-perfect trade-off between accuracy and performance in single-core VPs. No events are missed, while at the same time computation is not hindered by an undersized quantum. However, a downside of the dynamic quantum approach can surface in multi-core VPs. When a platform contains multiple event generating modules, the average quantum size may be diminished especially when one module generates events at a much higher frequency than the others. This leads to a deterioration in overall simulation performance that in some use-cases may be lower than the performance achieved by a well tuned static quantum.

4 EVENT RELEVANCE OPTIMIZATION

The dynamic quantum approach sets the quantum boundary at the time of the next scheduled SystemC event. Due to the global nature of the timed event queue, this is done regardless of whether receiving the event at this time is relevant for the correct functionality of the VP. There may be events that are internal to one module of the VP and the module whose quantum is limited by the event does not need to receive it at all. These events are called *irrelevant* events. For example, one component could contain a timer that overflows at certain points in simulation time. These events will be taken into account for all quantum computations, even though they are only relevant to the one component. This is depicted in Figure 2. It can be observed that the first event in the event queue limits the quanta of `CPU0` and `CPU1` albeit being irrelevant to either CPU. This unnecessarily decreases the overall performance of the simulation, because needless synchronization overhead is incurred.

To mitigate this issue we propose *Event Relevance Optimization (ERO)*. With ERO breaking the quantum is avoided when the responsible event is irrelevant to the component. This is done by taking event relevance into account during each quantum computation. An example is shown in Figure 3. Event 1 in the timed event queue is irrelevant to both CPUs, therefore it is not considered as a quantum boundary. The second event is only relevant to `CPU0`, which is why `CPU0`'s next quantum boundary is set to its notification time. For `CPU1` event 3 is the next relevant event.

To identify irrelevant events for ERO, a SystemC profiling tool was developed to trace process activations due to event notifications during a simulation run. Our tool incurred a runtime overhead of 214%, but only one profiling run is required. When the profiling step is finished, the gathered data is post-processed for analysis. During this post-processing an *event dependency graph* is generated. This is a weighed, directed graph whose nodes represent the SystemC events notified during simulation. The directed edges describe which SystemC process was started due to which event notification and which event was then notified by the process. There are two distinct event-process-event patterns that have to be taken into account. One pattern is the direct event notification described above, where a process is triggered by one event and notifies another event itself. The second pattern is an indirect event notification scenario, in which a process is triggered by a SystemC signal value change, that was in turn initiated by another process. In the latter case the signal value is updated by the SystemC kernel in response to another process's request instead of by the other process directly. After the construction of the event dependency graph is complete, it can be used for an automatic or manual analysis, e.g. by using graph visualization tools, to identify which events are irrelevant to a simulation component. These events can then be entered into an event denylist, which can then be supplied to the simulation.

It is important to keep in mind that for the ERO approach to be beneficial, the additional cost of checking event relevance has to be offset by the potential performance gain. In order to retain SystemC standard compatibility, an annotation of irrelevant events in the source code of the simulation components is not an option. Additionally, this is often entirely impossible because the source code of the simulation component may not be available, especially if it comes from a third party. Therefore, the dynamic quantum computation in the SystemC kernel is modified. Instead of setting

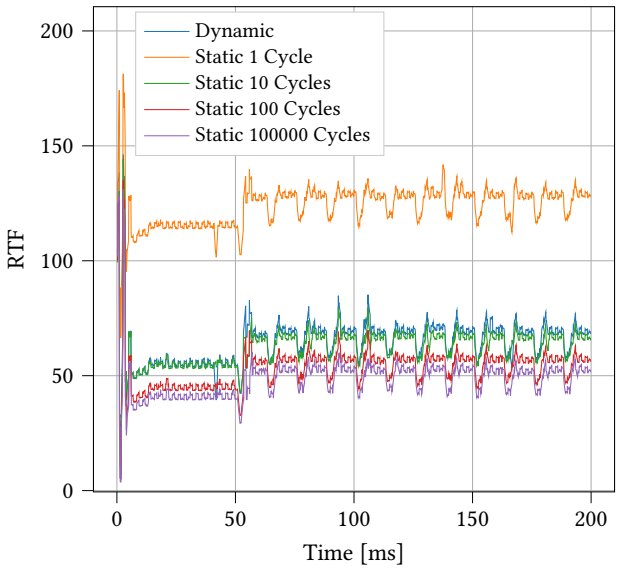

Figure 5: RTF_{smooth} over the entire simulation

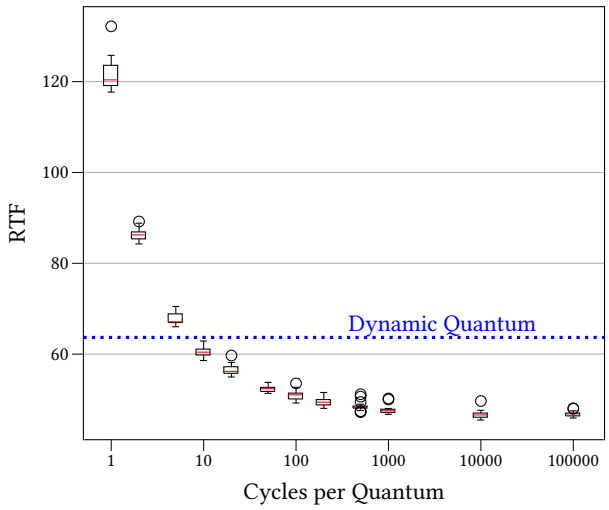

Figure 6: Average RTF over the entire simulation

the time of the next timed event as a quantum boundary, an additional relevance check is executed using the supplied denylist and the time of the next relevant event is used.

In the case study presented in the next section we show, that using our approach simulation performance can be improved significantly while still keeping nearly all the accuracy of the dynamic quantum approach.

5 CASE STUDY

The VP chosen for the case study is a register-accurate emulation of the Renesas RH850/E2x-FCC2 [3], a Microcontroller Unit (MCU) targeted for automotive tasks. An overview of the VP is provided in Figure 4. It can be observed that the platform consists of 6 CPUs, each attached to a complex bus infrastructure. The cores are subdivided into 3 clusters encompassing RAM and a flash memory storing the code for both processors of the cluster. A main system bus connects the clusters to a peripheral bus, which hosts 10 groups of peripherals. Among them, and highlighted by the red pattern, is a Bosch Generic Timer Module (GTM).

The GTM is a programmable, generic timer platform, which is used for engine control tasks. Once the CPU has initialized and configured the GTM for a specific task, it executes independently and keeps interactions to a minimum to reduce the CPU's load [6]. The GTM has a modular architecture which divides it into clusters. Inside each of these clusters execution is driven by a special-purpose, programmable core called Multi-Channel Sequencer (MCS). Each MCS unit manages up to 8 tasks or channels with a single pipelined data path. A built-in hardware scheduler selects the next task to be executed [6]. In this case study, the GTM is configured to contain 10 such clusters. Modeling this architecture and integrating it efficiently into the VP using SystemC TLM2.0 presents some challenges. Due to the GTM's highly modular structure, a high notification load is generated across the channels and many context switches occur between the clusters. To improve performance the MCS units employ temporal decoupling internally with a static quantum of 10 cycles. The quantum is deliberately kept small due to the module's strict timing requirements.

To focus on the interaction between cores and peripherals, a single-core, multiple-initiator task was chosen for evaluation. In this scenario `CPU0` executes in a temporally decoupled fashion while both cores of the cluster and multiple peripherals, such as reset and interrupt controller and the sub-modules within the GTM, can initiate transactions. The test software simulates an engine position management task, as outlined in [6]. For each cylinder in an engine, the crankshaft and camshaft position sensors generate signals from whose correlation different parameters, such as optimal cylinder ignition time, can be calculated. Instead of executing these calculations directly on `CPU0`, they are offloaded onto the GTM. In the evaluated scenario, the sensor signals are generated during simulation time and fed directly into the GTM whose MCS units are programmed to generate the engine position data. To communicate with the main CPU interrupts are used. The simulation task can be subdivided into two phases. First, an initialization phase occurs which lasts approximately 3 ms and is characterized by frequent transactions between the processor and various peripherals whose parameters are initialized. Afterwards, a steady-state phase begins in which the CPU enters an infinite control loop which is solely broken by sporadic interrupts triggered by peripherals, while the GTM processes the test inputs and calculates the engine position data. This latter phase continues indefinitely, but for evaluation purposes the simulation is stopped at the 200 ms mark.

6 EVALUATION

In this section static and dynamic quantum are evaluated against each other using the case study VP. Here, two evaluation criteria have to be taken into account, namely simulation performance and simulation accuracy. To measure the simulation performance, the

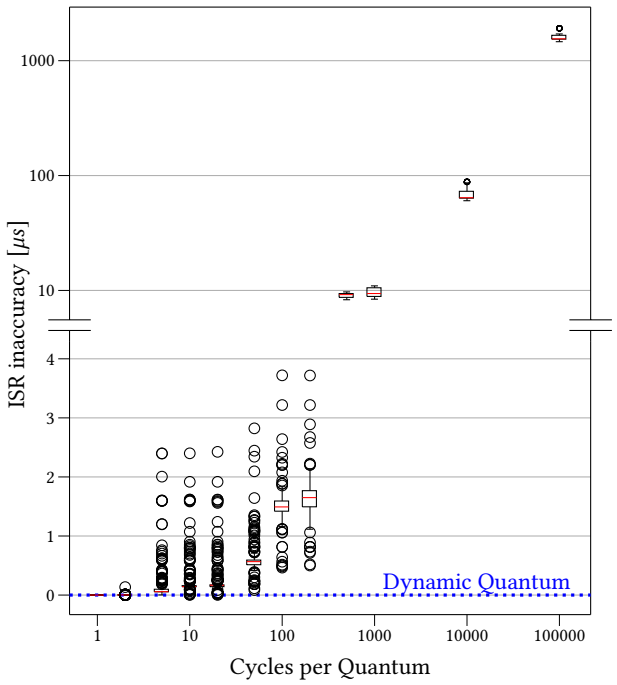

Figure 7: ISR timing inaccuracy

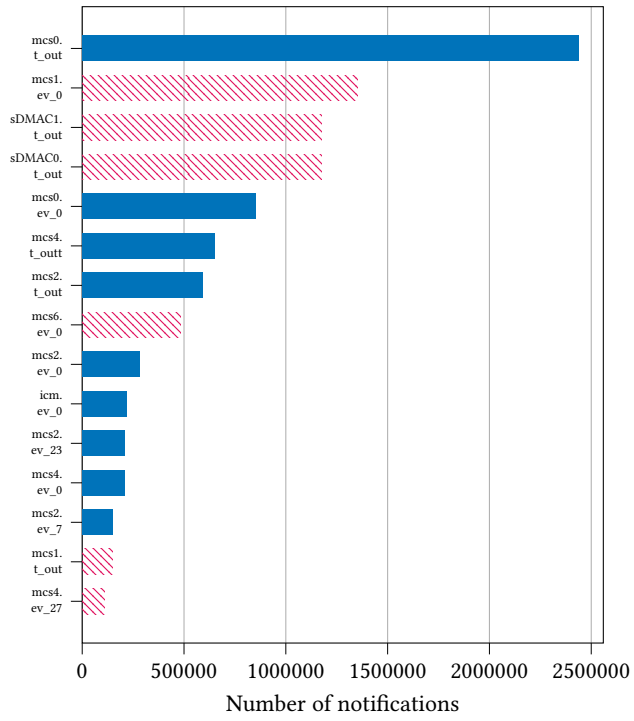

Figure 8: Most frequent quantum-limiting events

Real-Time Factor (RTF) is introduced as:

$$RTF[t_{\text{sim}}] = \frac{\text{Wall-clock time}}{\text{Simulation time}} = \frac{T_{\text{WC}}[0; t_{\text{sim}}]}{t_{\text{sim}}}$$

The RTF quantifies how much wall-clock time is required to simulate a period of simulation time. It is important to note that a lower RTF signifies higher simulation performance. The RTF measurement is undertaken 20 times over a complete simulation run after which the values are averaged. In addition to the averaged RTF, a smoothed RTF is introduced to analyze how the RTF changes over the duration of the simulation. The smoothed RTF is defined as:

$$RTF_{smooth}[t_{\text{sim}}] = \frac{T_{\text{WC}}[t_{\text{sim}} - 1ms; t_{\text{sim}}]}{1ms}$$

A granularity of 0.1 ms has been chosen for the RTF_{smooth} as a compromise between obtaining a precise result and avoiding large oscillations in the results.

6.1 Performance Evaluation

Figure 5 shows the smoothed RTF over the entire simulation. The initialization phase at the beginning can be clearly distinguished from the steady-state phase for the remainder of the simulation. During initialization the smoothed RTF is significantly higher than later in the simulation, because of the large number of context switches that occur between the components of the VP which are set up in this period. In this phase simulation performance is mostly determined by external components, which is why the frequency of context switches concerning the main core is less performance relevant than in later stages. This leads to the RTF curves for the different temporal decoupling schemes being relatively close to each other. Afterwards, the steady-state phase starts, which can be further subdivided into two stages. The former, ending roughly at the 53 ms mark, is characterized by low GTM activity. The RTF remains roughly constant during this phase. In the latter stage the RTF is dominated by the Interrupt Service Routine (ISR) triggered by the GTM, which is why a regular pattern can be observed. Peaks in the pattern correspond to ISR triggers and a speedup can be observed where no interrupts occur.

Overall, the dynamic quantum performs roughly on par with the static quantum with a quantum size of 10 cycles. This can be explained by the fact that the average quantum duration of the dynamic quantum amounts to circa 20 ns which corresponds to 8 cycles at the platform's clock frequency. If compared with the static quantum at a quantum size of 10 cycles, the dynamic quantum stands out growing to higher RTF peaks at interrupt trigger timestamps. This is due to the large number of ISR-related event notifications leading to frequent quantum breaks. For larger quantum sizes the static quantum outperforms the dynamic quantum.

The box plot depicted in Figure 6 shows the average RTF of the simulation for different quantum sizes for the static quantum scheme as well as for the dynamic quantum. It can be observed that the average RTF, which is at its highest in a single-stepping scenario, sharply decreases for small quantum sizes due to the drastic reduction in the number of context switches. Even a small quantum size of 2 cycles nearly halves the overhead needed for saving and restoring the CPU state after each quantum. The static quantum approach outperforms the dynamic quantum at a quantum size of 10 cycles. This can again be explained by the average quantum duration of the dynamic quantum of roughly 8 cycles. The value is

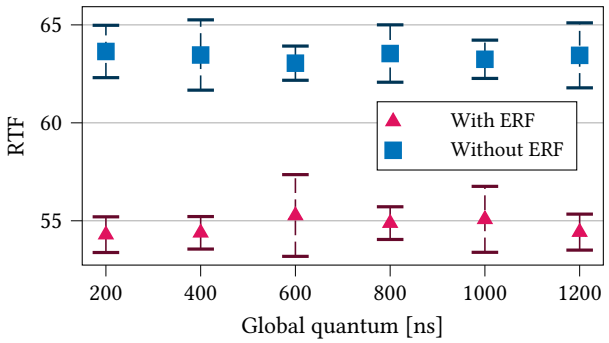

Figure 9: Average *RTF* with and without ERO

constrained by the large number of event notifications throughout the platform as outlined previously.

6.2 Accuracy Evaluation

Often an increase in simulation performance comes at the cost of reducing its accuracy. Hence, simulation accuracy is also analyzed in this case study. As explained previously, the GTM triggers the execution of an ISR. When executing the simulation using a static quantum with quantum sizes larger than 1 cycle, ISR execution is delayed because the signal is only processed by the CPU at the end of the quantum. This can be seen in the box plot in Figure 7, where this delay in the ISR execution is plotted for different static quantum sizes. The dynamic quantum is marked by the blue dotted line. It can be observed that there is next to no delay in ISR execution when using the dynamic quantum, because most events are processed as soon as they are notified, thus preserving near-perfect accuracy. However, starting at a static quantum size of 5 cycles, the introduced inaccuracy, in combination with the GTM being temporally decoupled internally, leads to a chain reaction due to delayed event handling which results in an average timing inaccuracy larger than the duration of a single quantum. In scenarios with 100 or more cycles per quantum, the average delay surpasses the 1 µs mark. Finally, with quantum sizes of 10.000 and 100.000 cycles per quantum respectively, the inaccuracies are disproportionately large to the point that the box plot in Figure 7 requires a logarithmic scale to accommodate larger values. These results confirm the expected timing behavior in a static quantum scenario.

In summary, the dynamic quantum approach preserves the perfect simulation accuracy of the single-stepping scenario while nearly delivering the RTF performance of the static quantum at a quantum size of 10 cycles. However, if some timing inaccuracy can be tolerated, the static quantum approach is able to outperform the dynamic quantum.

6.3 Event Relevance Optimization

As mentioned in the previous subsection, the suboptimal RTF values found when analyzing the dynamic quantum mostly stem from the large number of context switches due to the high event notification load in the VP. This is corroborated by the small average quantum size of circa 20 ns or 8 cycles, meaning that over the 200 ms simulation nearly 10.000.000 quanta have to be evaluated. In order to analyze which events restrict the average quantum size, the previously introduced SystemC profiler was deployed to trace event notifications during a simulation run. The generated traces show more than 11.000.000 quantum-limiting event notifications distributed over 3.860 events. The 15 most frequently notified events are shown in Figure 8. Despite the large number of different events limiting quantum size throughout the simulation, event notifications are not evenly distributed among them. Out of the 3.860 events, 3.348 are only notified once and further 195 twice. The remaining 317 events account for 99.97% of the total event notification load, while the 10 most frequently notified events among those reported in Figure 8 make up 81.61% of the total.

To improve the performance of the dynamic quantum the ERO technique presented in Section 4 is deployed. For this, the relevance of the most frequently notified events shown in Figure 8 is analyzed using the constructed event dependency graph. Afterwards, an ERO event denylist is constructed which contains the irrelevant events for the quantum calculation of `CPU0`. These events are highlighted by the red pattern in Figure 8. To evaluate the performance of the ERO approach, the average RTF is used. Figure 9 shows the average RTF for different quantum sizes comparing the ERO approach to the unmodified dynamic quantum. Overall, a performance gain of about 14.32% is achieved. As seen in Figure 6, dynamic quantum with ERO performs similar to static quantum with quantum sizes between 20 and 50 cycles. Furthermore, ERO virtually retains the same degree of accuracy as the unmodified dynamic quantum approach with all but one of the ISRs being triggered at the exact same instant in simulation. Being able to achieve a significant performance improvement while retaining close to perfect accuracy shows the potential of our approach. Further analysis of the tracing data shows that by deploying ERO the number of context switches can be reduced by nearly 29%. The number of quantum-limiting event notifications is reduced by about 35.58%.

7 CONCLUSION

In this work a thorough analysis of the two major temporal decoupling schemes for VPs has been conducted, namely the static and the dynamic quantum. Both approaches have their merits depending on the use case. If simulation performance is paramount and accuracy negligible to a certain degree, the static quantum with an aptly chosen quantum size is acceptable. Yet, the definition of this quantum size is largely left open to the developer and usually requires a cumbersome fine-tuning process in order to find the optimal trade-off. In addition, the optimal quantum size may vary over the simulation duration, which the static quantum cannot address.

The dynamic quantum is an alternative to the static quantum, yielding perfect simulation accuracy by setting the quantum boundary at the time of the next event notification. This way no event can be missed or handled too late. In complex multi-core VPs the dynamic quantum may run into limitations, if one component generates event notifications at a very high rate. This limits the quantum size of all components, because all timed event notifications are taken into account for the dynamic quantum computation, thus leading to diminished simulation performance. To improve the performance of the dynamic quantum approach, we propose the

novel ERO method to omit irrelevant events in the quantum size computation for each simulation thread.

Both approaches were thoroughly analyzed using a state-of-the-art, industrial VP and realistic software workload. For the static quantum it was shown that simulation accuracy deteriorates quickly for larger quanta. The unmodified dynamic quantum performs comparably to the static quantum with a quantum of 10 cycles. This is explained by the average dynamic quantum size of 8 cycles, which is due to the high event notification load in the VP. To optimize the VP's performance-accuracy trade-off, ERO was deployed. This way, the performance of the dynamic quantum was increased by 14.32% on average, which is comparable to using a static quantum size between 20 and 50 cycles. Furthermore, our dynamic quantum with ERO retains near-perfect simulation accuracy.

In future work, a method for automatically finding irrelevant events should be devised. Initial investigations have shown that this is a nontrivial task, due to the complexity of relationships between events. One way to deal with this complexity could be a heuristic that applies ERO on events deemed suitable and executes the simulation to gauge the impact on simulation correctness. Alternatively, a probabilistic or statistical method might be deployed to estimate event relevance. Using these techniques would further simplify applying ERO in practice.

REFERENCES

[1] 2011. *ISO 26262: Road Vehicles : Functional Safety.* ISO.
[2] 2012. Standard SystemC Language Reference Manual. *IEEE Std 1666-2011* (2012).
[3] 2020. RH850E2x product specification. https://www.renesas.com/eu/en/products/microcontrollers-microprocessors/rh850/rh850e2x/rh850e2m.html.
[4] Combes et al. 2008. Relaxing synchronization in a parallel SystemC kernel. In *2008 ISPA*. IEEE.
[5] Gläser et al. 2015. Temporal decoupling with error-bounded predictive quantum control. In *2015 Forum on Specification and Design Languages (FDL)*.
[6] Gowda et al. 2017. Exploring the potential of a Multi channel sequencer (MCS) in a next generation GTM-IP using virtual prototypes. In *2017 RTEICT*.
[7] Jung et al. 2019. Speculative Temporal Decoupling Using fork(). In *2019 DATE*.
[8] Rachuj et al. 2019. A Generic Functional Simulation of Heterogeneous Systems. In *International Conference on Architecture of Computing Systems*. Springer.
[9] Schumacher et al. 2010. parSC: Synchronous parallel Systemc simulation on multi-core host architectures. In *2010 CODES+ ISSS*. IEEE/ACM/IFIP.
[10] Schmidt et al. 2017. Exploiting thread and data level parallelism for ultimate parallel SystemC simulation. In *2017 DAC*.
[11] Weinstock et al. 2014. Time-decoupled parallel SystemC simulation. In *2014 DATE*. IEEE.
[12] Weinstock et al. 2016. SystemC-link: Parallel SystemC simulation using time-decoupled segments. In *2016 DATE*. IEEE.