Chapter 10
Compilation for Real-Time Systems a Decade After PREDATOR

Heiko Falk, Shashank Jadhav, Arno Luppold, Kateryna Muts, Dominic Oehlert, Nina Piontek, and Mikko Roth

10.1 Introduction

PREDATOR was a collaborative research project running from February 2008 until January 2011 that was funded by the European 7th Framework Programme under the lead of Reinhard Wilhelm, Saarland University, Germany. It was concerned *"with embedded systems that are characterized by efficiency requirements on the one hand and worst-case constraints on the other. [. . .] Embedded systems with critical constraints need off-line guarantees for the satisfaction of these constraints. Unfortunately, it can be observed that in computer system design, the gap between average-case and worst-case behavior increases rapidly. This entails a decreasing precision of performance analysis results."* Therefore, PREDATOR proposed *"a new research and design discipline that looks at predictability and efficiency in a synergistic manner and that involves all levels of abstraction and implementation in embedded system design"* [6, 34].

These different abstraction levels were reflected by the project's scientific work packages. WP1 (led by Luca Benini, University of Bologna, Italy) dealt with predictable and efficient hardware architectures. Both functional and power models of a predictable architecture were developed and their sensitivity to architectural parameters that influence predictability and their costs were analyzed. WP2 (lead: Peter Marwedel, University of Dortmund, Germany) focused on compiler and code generation techniques for a single application task. Here, optimizations that are aware of hard real-time constraints and of Worst-Case Execution Times (WCET) were proposed; multi-objective trade-offs between real-time guarantees and energy consumption or code size were envisioned. WP3 was led by Giorgio Buttazzo

H. Falk (✉) · S. Jadhav · A. Luppold · K. Muts · D. Oehlert · N. Piontek · M. Roth
Institute of Embedded Systems, Hamburg University of Technology (TUHH), Hamburg, Germany
e-mail: Heiko.Falk@tuhh.de

J.-J. Chen (ed.), *A Journey of Embedded and Cyber-Physical Systems*,
https://doi.org/10.1007/978-3-030-47487-4_10

(Scuola Superiore Sant'Anna, Pisa, Italy) and targeted the coordination of multiple application tasks. Off-line and online coordination techniques were investigated such that guarantees on tasks' response times were derived under simultaneous optimization of resource usage. In the context of WP4 (lead: Lothar Thiele, ETH Zürich, Switzerland), distributed embedded systems were considered and the modular analysis of Multi-Processor Systems on Chip (MPSoC) with respect to performance and predictability was investigated. Finally, cross-layer aspects of the design and analysis of predictable and efficiency were considered in WP5 (lead: Reinhard Wilhelm).

Overall, PREDATOR was a high-quality collaborative effort that produced many seminal results in the field of designing predictable and efficient hardware and software architectures. On the occasion of Peter Marwedel's 70th anniversary, this article surveys the results in the area of compilers for real-time systems that have been achieved under his leadership within PREDATOR. The foundational character of this project is highlighted by providing an overview over code optimizations and analyses proposed in the past decade since PREDATOR was executed. These recent works directly base on challenges identified during and on results produced by PREDATOR.

Section 10.2 puts the state-of-the-art in compilation for real-time systems by the end of PREDATOR in a nutshell. Recent developments that integrate task coordination into compiler optimizations are described in Sect. 10.3. The combination of system-level analysis and code generation techniques for parallel multi-core systems is the subject of Sect. 10.4. Section 10.5 discusses multi-objective compiler optimizations that are able to adhere to real-time constraints, and Sect. 10.6 concludes this article and provides an outlook over future work.

10.2 Challenges and State-of-the-Art in WCET-Aware Compilation During PREDATOR

A program's WCET stands for its maximal possible execution time, irrespective of possible input data and of initial states of the hardware architecture. For the design of hard real-time systems, the WCET is a critical design parameter, since it allows to reason about whether a program always meets its deadline or not. However, the exact computation of a program's WCET is infeasible in general so that conservative WCET estimates are used instead. In the domain of hard real-time systems, such WCET estimates are usually produced by static timing analysis tools, e.g., aiT [1]. During PREDATOR's single-task activities carried out within Work Package WP2, such a timing analyzer was tightly integrated into a compiler framework. This allowed the compiler to perform WCET analyses in a fully automated fashion during code generation. The WCET data gathered this way constitutes a precise worst-case timing model inside the compiler which contrasts sharply with standard compilers that focus on average-case scenarios and that do not feature any timing models at all. The resulting WCET-aware C Compiler WCC [8] finally exploits this precise timing model in dedicated WCET-aware, single-task code optimizations.

However, modern real-time systems do not consist of only a single task—they are multi-task systems instead where tasks are preempted and scheduled according to an operating system's scheduling policy. Thus, the design of a timing predictable multi-task system includes the consideration of all tasks' end-to-end latencies including blocking times due to preemptions, i.e., the tasks' Worst-Case Response Times (WCRT). Based on the tasks' WCRTs, a subsequent schedulability analysis can be used to determine whether all tasks definitely meet their respective deadlines. Since WCETs are characterized by the behavior of machine code for a given processor architecture, and since WCRTs and schedulability analyses rely on given WCET values and mostly depend on task-level scheduling properties, there is a natural link between compilers and operating systems: the former generate the machine code that the latter have to schedule. This link was already identified during PREDATOR:

Challenge #1

"The compiler [. . .] will apply optimizations not for each individual task in isolation, but will consider all tasks of the entire system in a holistic view. Furthermore, it is planned to take the individual scheduling policies [. . .] into account" [5].

Plazar et al. proposed a software-based cache partitioning for real-time multi-task systems [29]. Cache partitioning is able to make the behavior of instruction caches more predictable, since each task of a system is assigned to a unique cache partition. The tasks in such a system can only evict cache lines residing in the partition they are assigned to. As a consequence, multiple tasks do not interfere with each other any longer w.r.t. the cache during context switches. This allows to apply static WCET analysis for each individual task of the system in isolation. The overall WCET of a multi-task system using partitioned caches is then composed of the WCETs of the single tasks given a certain partition size, plus the overhead required for scheduling and context switching. Until the completion of PREDATOR, an integration of schedulability analyses and a consideration of individual scheduling policies during compilation could not be realized due to a shortage of time.

In the context of performance analysis for massively parallel multi-core architectures, PREDATOR proposed a modular approach where a WCET analysis is performed for each application per individual processor core in isolation. By exploiting how often each core accesses the shared bus that connects all cores in a given MPSoC architecture, the additional timing interference that each processor core exhibits due to temporarily blocked bus accesses is estimated. According to PREDATOR's design rules for predictable architectures [38], TDMA-arbitrated shared buses were considered during modular performance analysis. In the end, upper timing bounds of all applications running on all processor cores are derived in a modular fashion which allows to reason about schedulability for such parallel multi-core systems [30].

Various execution models for the applications running on such an MPSoC architecture were considered. In the so-called Dedicated Access Model, applications are structured into three distinct phases: acquisition, execution, and replication. Only during the first and the latter, a task is allowed to access the shared bus in order to fetch input data or to write back computed results, resp. Since the main execution phase of a task is free of shared bus accesses, it cannot suffer from delays induced by other cores which allows for a very precise timing analysis. In the General Access Model, accesses to the shared bus can happen anytime during acquisition, replication, and execution. Thus, a timing analysis becomes more pessimistic here [31].

As the precision of timing analysis for MPSoCs thus strongly depends on the execution behavior of tasks, mechanisms enforcing well-suited and predictable access patterns to shared buses would be advantageous.

Challenge #2

"One new possibility to reduce the effect of (timing) interactions [. . .] is the use of traffic shapers. It is an open problem to include these units into a system-wide performance analysis that considers computation and communication resources" [5].

However, PREDATOR did not come up with approaches addressing this challenge.

PREDATOR explicitly considered the trade-off between predictability where hard constraints on a system's resource usage must be met versus the efficiency of a system in the average case.

Challenge #3

"We will develop models capturing various optimization objectives within the compiler, e.g. code size or energy dissipation [. . .]. Novel optimization strategies are designed in order to minimize an objective other than WCET, under simultaneous adherence to real-time constraints" [5].

Since the WCC compiler featured a detailed WCET timing model right from the project start, and since modeling code size at the assembly code level is trivial from a compiler's point of view, it was obvious to consider trade-offs between these two objectives in the beginning. For this purpose, simple heuristics for the optimization Procedure Cloning were proposed where WCETs were minimized as long as the resulting code sizes did not exceed a user-provided threshold [19]. Later, WCC was coupled with an instruction set simulator allowing to perform dynamic profiling during compilation. Furthermore, data from an instruction-level energy model [32] was also integrated. This way, the compiler was able to simultaneously model WCET, code size, ACET, and energy consumption of generated machine code.

These models were used to determine Pareto-optimal sequences of compiler optimizations. It is a well-known problem that the order in which a compiler applies its optimizations can have a significant impact on the quality of the finally generated code. In the context of PREDATOR, a stochastic evolutionary multi-objective algorithm [20, 21] found optimization sequences that trade pairs of objectives, i.e., ⟨WCET, ACET⟩ and ⟨WCET, code size⟩, resp. True multi-objective code optimizations that inherently model and consider different criteria at the same time during code generation have, however, not been investigated in depth during PREDATOR.

10.3 Integration of Task Coordination into WCET-Aware Compilation

Many architectures are equipped with fully software-controllable secondary memories. These are memories that are tightly integrated with the processor to achieve the best possible performance. These Scratchpad Memories (SPMs) can be accessed directly and are therefore in general well-suited for optimizations regarding energy consumption and execution times.

SPMs turned out to be ideal for WCET-centric optimizations, since their timing is fully predictable. The WCC compiler exploits SPMs for WCET minimization by placing assorted parts of a program into a scratchpad memory. During PREDATOR, an Integer-Linear Program (ILP) originally proposed by Suhendra et al. [33] was extended towards a single-task SPM allocation where binary decision variables x_i are used per basic block b_i. b_i is moved from main memory onto the scratchpad memory if x_i equals 1. The overall goal of this ILP is to find an assignment of values to the variables x_i such that the resulting SPM allocation leads to the minimal WCET of the whole task. Constraints are added to the ILP that model the task's internal program structure. For each basic block b_i and each successor b_{succ} of b_i in the task's Control Flow Graph (CFG), a constraint is set up bounding the WCET c_i of b_i:

$$c_i \geq c_{succ} + cost_{i,main\_mem} - gain_i * x_i \tag{10.1}$$

This constraint states that the WCET c_i of a path starting in b_i must be larger than the WCET c_{succ} of any of the successors of b_i, plus the contribution of b_i to the WCET itself with b_i located in main memory ($cost_{i,main\_mem}$), minus the potential gain when moving b_i from main memory onto the scratchpad memory. Additional constraints in the ILP model loops and function calls. The limited available capacity of the SPM is considered as well as the additional overhead due to long-distance jumps from the main memory to the SPM or back. In the end, the WCET of an entire task is represented in the ILP model by a variable $c^{entry}_{\mathtt{main}}$ which models the WCET of the path starting at the task's entry point, i.e., at its `main` function [7].

This basic ILP model turned out to be very powerful and flexible so that it served as the basis for the optimization of multi-task systems. For this purpose, all tasks of a multi-task application were modeled in the ILP as described above. As a consequence, the ILP variables c_j associated with the entry points of the tasks τ_j describe a safe upper bound of the tasks' WCETs. An early work [22] towards PREDATOR's Challenge #1 on optimization of multi-task systems under consideration of scheduling policies integrated Joseph's schedulability analysis [15] into this multi-task ILP.

For priority-based scheduling, a task τ_j's WCRT r_j is the maximum possible time interval between the activation of a task and its end, including penalties due to preemptions by higher-priority tasks. The tasks' WCRTs are computed as follows:

$$r_j = c_j + \sum_{h=0}^{j-1} \left\lceil \frac{r_j}{T_h} \right\rceil * c_h \tag{10.2}$$

Eq. (10.2) accumulates the net WCET c_j of task τ_j and the penalties due to tasks $\tau_0, \ldots, \tau_{j-1}$ having higher priority than τ_j. Each such high-priority task τ_h preempts τ_j a total of $\left\lceil \frac{r_j}{T_h} \right\rceil$ times where T_h denotes a task's period. For each preemption of τ_j by τ_h, the higher-priority task's WCET c_h is considered.

However, it is not straightforward to integrate Eq. (10.2) into an optimization's ILP, since both the WCETs c_h and the WCRTs r_j are ILP variables so that the multiplication of $\left\lceil \frac{r_j}{T_h} \right\rceil$ by c_h is infeasible. In order to solve this problem, an integer variable $p_{j,h}$ is added to the ILP for every combination of low- and high-priority tasks τ_j and τ_h, resp. $p_{j,h}$ denotes the timing penalty that is added to τ_j's WCRT due to preemptions by τ_h. Using these variables, Eq. (10.2) can be rewritten to:

$$r_j = c_j + \sum_{h=0}^{j-1} p_{j,h} \tag{10.3}$$

In order to model $p_{j,h}$, the following linearization scheme is applied: If r_j is lower than or equal to τ_h's period T_h, τ_j can be preempted at most once by τ_h, thus leading to $p_{j,h} = 1 * c_h$. If r_j is greater than T_h but lower than or equal to $2 * T_h$, $p_{j,h} = 2 * c_h$ results, etc. In general, it has been proven that

Theorem 10.1 *If τ_j is preempted at least N times by τ_h, then $p_{j,h} \geq (N + 1) * c_h$ must hold.*

Such so-called conditional constraints can efficiently be translated into ILP in Eq. [25]. A natural upper bound for the number N of preemptions of τ_j by τ_h is $\left\lceil \frac{D_j}{T_h} \right\rceil$ where D_j denotes task τ_j's deadline. Thus, the conditional constraints from Theorem 10.1 are added to the ILP for all values of N with $0 \leq N \leq \left\lceil \frac{D_j}{T_h} \right\rceil - 1$ and for all pairs of low- and high-priority tasks τ_j and τ_h, resp. Finally, the schedulability of the entire multi-task set is ensured during this ILP-based optimization by adding constraints

$$r_j \leq D_j \tag{10.4}$$

such that the WCRT of each task τ_j must be within its respective deadline.

While this work is a first step towards schedulability-aware compiler optimization, it suffers from a couple of limitations: First, the task model only supports fixed-priority scheduling and periodic tasks. Second, preemption costs due to the execution of an actual scheduler and context switching overheads are not considered. Finally, the number of constraints of the ILP proposed in [22] grows quadratically with the size of the task set, and it depends on the actual values for tasks' deadlines and periods.

The consideration of Liu and Layland's schedulability test [18] helped to overcome the limitation to fixed priorities:

$$u = \sum_j \frac{c_j}{T_j} \leq 1 \tag{10.5}$$

Eq. (10.5) states that a system that is scheduled with dynamic priorities using Earliest Deadline First (EDF) is schedulable if and only if the system load u is less than or equal to 1. Due to the already linear nature of Eq. (10.5), it is easy to integrate this schedulability test into an ILP [22].

The relaxation of strictly periodic task sets required to use an event-based task model supporting arbitrary task activation patterns and deadlines [24]. For this purpose, the ILP described above has been extended by support for density and interval functions η and ϵ, resp., as originally proposed by Gresser [10] and later taken up by Albers et al. [2]. In this approach, an arbitrary kind of task activation pattern can be characterized by the density function η that denotes the maximum number of events (i.e., task activations) in some time interval Δt. The interval function ϵ models the inverse behavior and returns the minimal time interval Δt in which n tasks are activated. This task model provides a high flexibility so that periodical multi-task systems, periodical systems with jitter or bursts, or systems with fully arbitrary task activations can be modeled in the optimization's ILP.

The consideration of an actual scheduler's overhead for context switching can be added to the ILP-based framework described above by introducing an implicit task τ_0 with the highest priority into the multi-task system. τ_0 represents the periodically executed scheduler, and by considering an actual scheduler's WCET c_0 and its period T_0, it can smoothly be integrated into the optimization framework [23].

As an alternative to Joseph's schedulability test (cf. Eq. (10.2)), Baruah proposed the so-called processor demand test [3]. It states that a multi-task system is schedulable if and only if the amount of *required* computation time is less than or equal to the amount of *available* computation time:

$$\Delta t \geq \sum_{\forall \tau_j} \left[\eta_j \left(\Delta t - D_j \right) * \left(c_j + o_j \right) \right] \tag{10.6}$$

According to the event-based task model described above, Δt denotes one time interval to be analyzed. $\eta_j(\Delta t - D_j)$ returns the number of activations of task τ_j that happen within Δt and that must be finished before the deadline D_j. Each task activation is multiplied by the task's respective maximum computational demand, i.e., its WCET plus additional preemption overheads o_j.

Since Eq. (10.6) is linear, it can directly be added to our multi-tasking ILP model for each task τ_j. This schedulability test has to be modeled for all possible time intervals Δt. The maximal interval to be considered is regularly given by the task set's hyperperiod. Checking all possible intervals Δt up to the hyperperiod is practically infeasible. Fortunately, task preemptions can only occur if a new task is ready for execution for many real-life scheduling policies like, e.g., EDF. Thus, the schedulability test from Eq. (10.6) has to be modeled in the ILP only at the points of discontinuity of the task set's density functions η. Finally, one constraint needs to be added that ensures that the system's overall load due to periodically repeating task activations stays below 100%. It is also possible to extend this approach towards fixed-priority scheduling, and the resulting ILP model grows only linearly with the number of events that have to be analyzed, in contrast to the quadratic nature inherent to [22, 24].

Figure 10.1 shows the effect of our ILP-based multi-task SPM allocation on schedulability of task sets featuring 8 tasks. We randomly selected 20 different task sets from TACLeBench [9]. Task periods were also randomly determined using UUniFast [4] and adjusted [39]. For each task set, periods were assembled such that the entire system has an approximate initial load of 0.8, 1.0, …, 2.2 i.e., 8 different system loads are evaluated per task set. Task deadlines were chosen uniform randomly between 0.8 and 1.2 times the task's period. Furthermore, a jitter of up to 1% of each task's period was chosen uniform randomly. Our evaluation considered an ARM-based architecture with access latencies for main memory and SPM of 6 and 1 clock cycles, resp. The scratchpad size was set to 40% of each task set's total size.

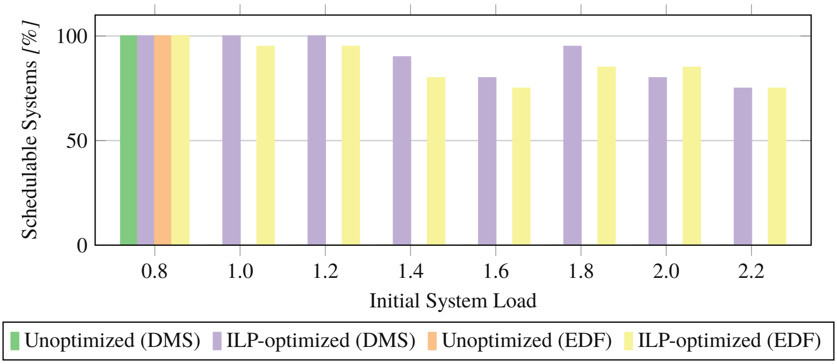


Fig. 10.1 Evaluation of schedulability-aware SPM allocation for 8 tasks

Figure 10.1 shows the schedulability of the task sets for the given initial system loads, using Deadline-Monotonic Scheduling (DMS) and Earliest Deadline First (EDF). The green and orange bars show the percentage of schedulable systems without any optimization applied, while the purple and yellow bars represent the schedulability after our ILP-based multi-task SPM allocation.

For an initial system load of 0.8, all task sets are schedulable, irrespective of the considered scheduling policy or whether the SPM allocation was applied or not. This is not surprising, since the considered systems feature sufficient idle times so that valid schedules are always found. The situation changes when considering higher initial system loads that range from 1.0 up to 2.2. In these scenarios, no task set was schedulable in an unoptimized state where the scratchpad memories were not used at all. However, our multi-task optimization is able to turn the vast majority of initially unschedulable task sets schedulable. For DMS scheduling, our ILP-based optimization achieves rates of schedulable task sets ranging from 100% (initial system loads of 1.0 and 1.2) to still 75% for an initial system load of 2.2. For EDF scheduling, the percentages of finally schedulable task sets are slightly smaller—they range from 95% (initial system load of 1.0) to 75% again. The time required to solve our ILPs is moderate. The whole compilation, analysis, and optimization process using a modern ILP solver like, e.g., `gurobi` required less than 6 CPU minutes on average over all considered task sets.

10.4 Analysis and Optimization of Multi-Processor Systems on Chip

To address PREDATOR Challenge #2 on analyzing and shaping the communication traffic for MPSoC architectures, it is important to understand when events happen in a multi-core architecture which potentially influences the cores' timing behavior. For this purpose, modular performance analyses use so-called request functions α which are very similar to the density function η from Sect. 10.3. In the context of MPSoCs, however, such functions characterize how often a processor core requests the shared bus of a multi-core architecture within a certain interval of time. Usually, such functions are provided at a very abstract level assuming execution models consisting of, e.g., the aforementioned acquisition, execution, and replication phases. For a precise analysis when each core attempts to access a shared hardware resource, it is, therefore, beneficial to extract request functions at the machine code level [14, 27].

For a precise and tight MPSoC performance analysis, both lower and upper bounds of resource requests are generated. Positions within the machine code executed on the different cores are identified where timing-relevant requests are generated, i.e., where shared hardware resources are accessed. Based on the code's Control Flow Graph (CFG), all possible sub-paths inside the code that feature these identified positions have to be considered. For this purpose, the well-known Implicit

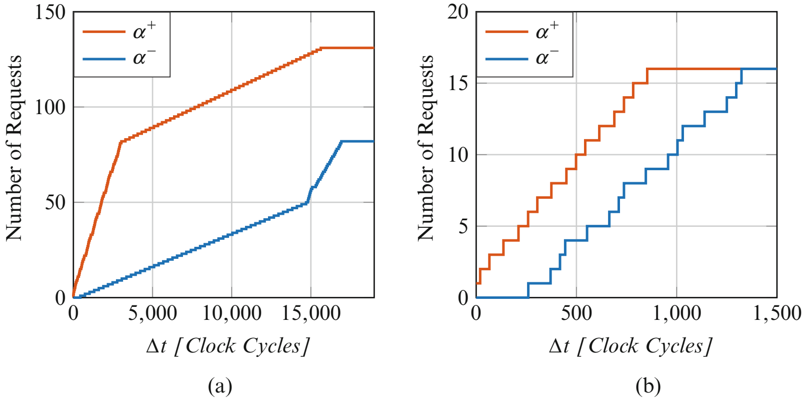


Fig. 10.2 Extracted request functions for selected benchmarks. (**a**) `Compressdata`. (**b**) `binarysearch`

Path Enumeration Technique (IPET) [17] has been modified to find the maximum number of requests potentially occurring in a given time interval along any path of a program. An algorithm has been proposed [27] that provides bounds on the number of requests for time intervals Δt of a program's runtime under consideration of all possible paths inside the CFG. This algorithm can be parameterized to trade precision of the generated request functions versus required execution time by varying the number of sampling points, i.e., the granularity of time intervals Δt considered by the algorithm.

Examples of lower (α^-) and upper (α^+) request functions generated for two selected benchmarks `compressdata` and `binarysearch` are shown in Fig. 10.2. The vertical distance between the lower and upper functions shows the variation of the number of produced requests. For example, `compressdata` can terminate with solely 82 shared bus accesses in total, or with up to 131. For `binarysearch`, both the lower and upper request functions converge to a common value, since each possible path through the program's code covers an identical number of bus requests. Only the points in time when these events occur differ.

Figure 10.3 shows the influence of the number of considered sampling points on the precision of the upper request function α^+ of `compressdata`. The finest-possible granularity, i.e., $\Delta t = 1$ clock cycle, leads to 131 samples in total and to a very smooth and precise result. When reducing the granularity such that only 50 samples are considered, the resulting request function has a clearly visible stepwise shape. However, the resulting function for 50 samples always dominates the most precise function so that no unsafe results are produced. For the highest precision with 131 samples, our algorithm requires 48 CPU seconds. In contrast, the time required to generate the request function for `compressdata` decreases down to 10 CPU seconds if 50 sampling points are considered.

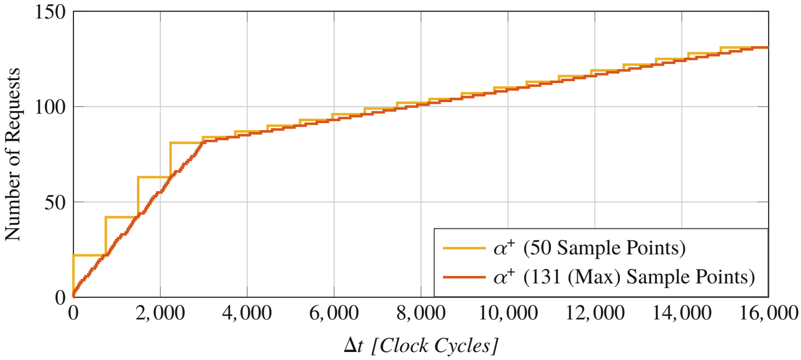


Fig. 10.3 Request functions for `compressdata` with different precision levels

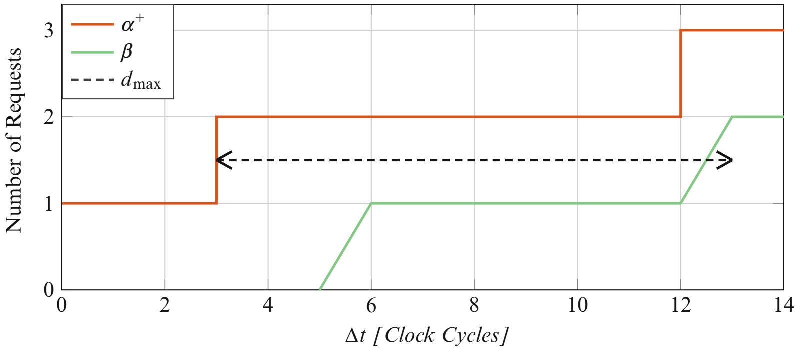


Fig. 10.4 Request functions α and delivery functions β

While request functions α denote the resource demand of a task w.r.t. shared bus accesses, so-called delivery functions β model the available capacity of a shared hardware resource during modular performance analysis [12, 35]. The relationship between both types of functions is illustrated in Fig. 10.4. The maximal horizontal distance between α^+ and β represents the maximum delay $d_{\max}$ a task exhibits due to blocked shared bus requests. In the figure, a task requests 2 bus accesses during interval lengths of 3 clock cycles. However, the bus can deliver the desired capacity only within 13 clock cycles. Thus, a blocking time of 10 clock cycles results from Fig. 10.4.

If a compiler could modify the generated code such that a task's request function is shifted towards the rightmost end of Fig. 10.4, its blocking time gets reduced which in turn probably decreases WCRTs and improves schedulability for the entire MPSoC system. This approach was investigated by a Master's Thesis [28] where instruction scheduling was exploited. Locally within basic blocks, those instructions requesting shared bus accesses were postponed by scheduling independent instructions in front of them. If this succeeds for all program paths of a given length

Δt (e.g., for $\Delta t = 3$ in Fig. 10.4), then the request functions are actually shifted as intended. This work revealed that compilers can be enabled to systematically reduce blocking times this way. For MPSoC task sets generated from the MRTC [11] and UTDSP [37] benchmark collections, blocking time reductions of up to 22.5% were reported. A solely local rescheduling of instructions, however, suffers from the inherent limitation that there is not too much potential for postponing shared bus accesses within a single basic block. Thus, maximal WCRT reductions of only up to 7.3% were achieved.

This basic idea to reshape bus requests at the code level is also pursued in currently ongoing work. By transforming the behavior of a task, its request function is modified such that its traffic will match a required profile. This is done by inserting additional machine instructions into the code, i.e., `NOP`s. Therefore, this approach does not rely on specific hardware or on operating systems that realize traffic shaping. Instead, the notion of code-inherent traffic shaping is introduced. If the places where to insert such additional instructions in a task's CFG are carefully chosen, parts of its request function that do not fit to a given access profile can be shaped systematically, even without necessarily increasing the task's WCET. For this purpose, two shaping algorithms using a greedy heuristic and an evolutionary algorithm have been designed which support various kinds of Leaky Bucket shapers [36].

The effectiveness of code-inherent shaping is depicted in Fig. 10.5 by means of MRTC's `select` benchmark. Based on a Leaky Bucket that generates a stepwise shaping profile, a delivery function β is assumed such that only half of the requests originally issued by the task within 1000 clock cycles can be fulfilled. It can be seen that the systematic insertion of a total of 408 `NOP` instructions results in a shaped request function that always stays below this delivery function. For this particular `select` task, its WCET increases from originally 36,019 clock cycles up to 50,317 clock cycles. While this WCET increase by 40% seems disadvantageous at a first glance, it is absolutely acceptable if the task still meets its deadline and if the shaped request function enables schedulability of the entire MPSoC task set.

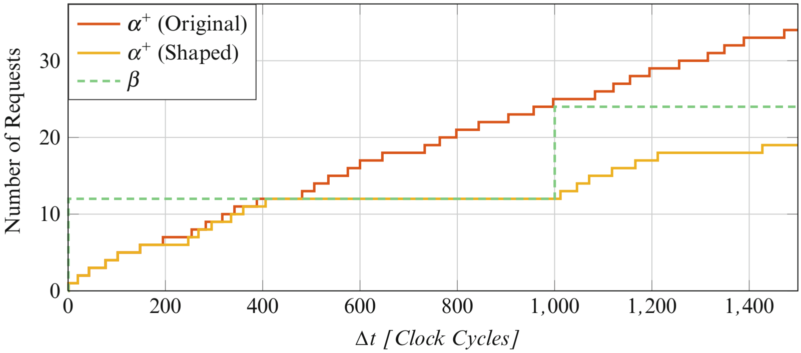


Fig. 10.5 Traffic shaping of `select` with $\beta(\Delta t)$ being 50% of $\alpha(\Delta t)$ for $\Delta t = 1000$

10.5 Multi-Objective Compiler Optimizations Under Real-Time Constraints

The simultaneous consideration of multiple optimization objectives by a compiler according to PREDATOR Challenge #3 can, to some extent, already be achieved using ILP-based techniques, even though ILPs only allow for one objective function to be maximized or minimized. PREDATOR's distinction between efficiency requirements on the one hand and worst-case constraints on the other hand naturally suggests to model critical constraints that must always be fulfilled as inequations in an ILP. Efficiency requirements are then modeled by an ILP's objective function and get optimized in addition to the satisfaction of critical constraints. This way, it is rather straightforward to turn the multi-task scratchpad memory allocation described in Sect. 10.3 into a multi-objective WCET-, schedulability- and energy-aware optimization.

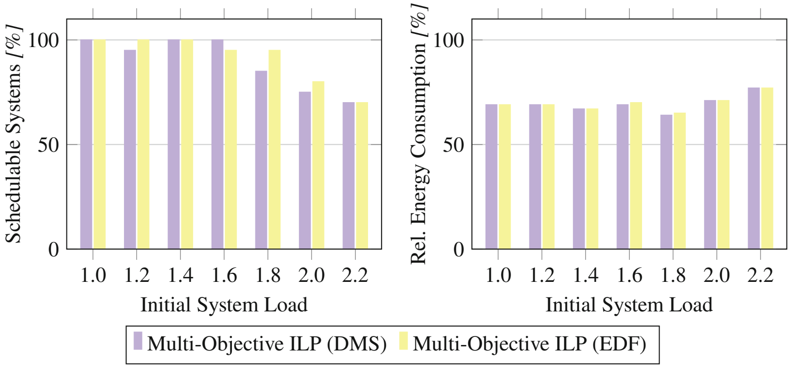
The schedulability tests from Eq. (10.4) or (10.6) are mandatory constraints in the SPM allocation's ILP model. Using an energy model like, e.g., [32], the energy consumption e_i of each basic block b_i can be characterized in dependence of the ILP's binary decision variables x_i. By combining these block-level energy values with profiling-based information about the blocks' execution frequencies, the overall energy consumption e_j of task τ_j can be modeled. Multiplying these task-level energy values with the tasks' activation functions η_j (cf. Sect. 10.3) over the entire task set's hyperperiod H yields an expression that models the energy dissipation of the complete multi-task system and that thus can be minimized under simultaneous adherence to the ILP's schedulability constraints:

$$\min \sum_{j} \eta_j\,(H) * e_j \tag{10.7}$$

Evaluation results for randomly generated sets of 6 tasks are depicted in Fig. 10.6, the experimental setup is the same as described in Sect. 10.3. Figure 10.6a shows the task sets' schedulability for their respective initial system loads, again using DMS and EDF scheduling. As can be seen, the multi-objective ILP is able to turn more than 95% of all task sets schedulable for initial system loads of up to 1.6. For higher initial loads, schedulability was still achieved for more than 70% of all task sets.

Simultaneously, considerable energy reductions compared to systems that do not use the SPM were achieved, cf. Fig. 10.6b. For initial system loads of up to 1.8, the task sets' energy dissipation was reduced down to less than 70%. For higher initial system loads, the resulting energy consumption still ranges from 71 to 77%.

Another common additional optimization goal is to meet code size requirements. Code compression might be used to meet code size constraints in embedded systems. However, the performance overhead of such techniques might be critical for real-time systems that must adhere to strict timing constraints. In the context of PREDATOR Challenge #3, we thus recently considered compiler-based code compression for hard real-time systems for the very first time [26]. This approach



(a)

(b)

Fig. 10.6 Evaluation of multi-objective schedulability- and energy-aware SPM allocation for 6 tasks. (**a**) Schedulability. (**b**) Energy consumption

exploits lossless asymmetric compression algorithms [13] where a computationally demanding and highly effective code compression is performed at compile time, while the decompression is computationally lightweight so that it is feasible to perform it at runtime.

In the proposed approach, complete binary executable functions are selected and compressed by the WCC compiler and the resulting bit stream is added to the executable code produced by the compiler. Furthermore, the executable is extended by specifically tailored code for the decompression of the selected functions. Upon execution of a program optimized this way, all compressed functions are decompressed in one go during the program's start. For this purpose, a processor's scratchpad memory is used as a buffer that finally holds all decompressed functions. These functions are then directly executed from the SPM.

This approach trades code size reductions due to the selection of functions to be compressed with the decompression overheads in terms of WCET which should be as small as possible. For this purpose, an ILP is proposed whose binary decision variables x_i encode whether function f_i is compressed or not.

For each function f_i that might be compressed, its original, uncompressed code size S_i^{orig} and its Worst-Case Execution Time C_i^{orig} are pre-computed. Assuming that f_i would be compressed, the corresponding values S_i^{comp} and C_i^{comp} can also be pre-determined. For the WCET analysis of a potentially compressed function f_i, the decompression routine is added by the compiler, and the loops therein are precisely annotated with upper iteration bounds for the decompression of the currently considered function f_i in order to support the WCET analyzer aiT. Based on this data, the impact of f_i's compression on the entire program's code size ΔS_i and Worst-Case Execution Time ΔC_i can be expressed in the ILP.

ILP constraints ensure that the decompressed functions fit in the available SPM, that the entire program never gets larger due to the inserted decompression routine, and that the WCET increases of all functions always stay below a user-provided

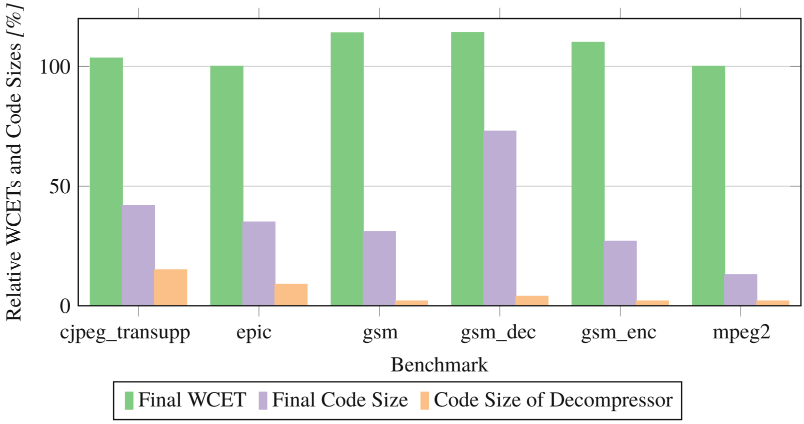


Fig. 10.7 Evaluation of compiler-based WCET-aware code compression for MediaBench

threshold ΔC^{limit}. Under these constraints, the ILP finally minimizes the entire program's code size by selecting appropriate functions f_i for compression.

For six large-sized benchmarks from MediaBench [16], the effects of the proposed compiler-based code compression for an Infineon TriCore architecture are depicted in Fig. 10.7. For each considered benchmark, the diagram shows the resulting relative WCETs and code sizes, as well as the code size of the decompression routine added by the compiler. The 100% baseline of Fig. 10.7 denotes the WCETs and code sizes of the original, unoptimized benchmarks, resp. For the ILP-based selection of functions to be compressed, the threshold ΔC^{limit} was set to 0.5 so that maximum WCET increases by 50% were still accepted by the optimization.

As can be seen from Fig. 10.7, the finally obtained WCET increases are way below this user-provided upper bound. For `epic` and `mpeg2`, the WCETs degrade only marginally by 0.6% and 0.5%, resp. The WCETs of the other benchmarks increase between 3.5% and 14.1% only. In contrast to this, our approach achieves rather large code size reductions. After the optimization of `gsm_dec`, its executable occupies only 73% of its original memory space. For all other benchmarks, an even higher degree of compression was achieved that reduces code sizes by more than a half. This way, the code size of `cjpeg_transupp` was reduced to 42% of its original size, and a maximal reduction down to only 13% of the original code size was achieved for `mpeg2`. Finally, Fig. 10.7 shows that adding extra code to the generated binaries for the decompression routine is worthwhile, since this overhead is over-compensated by the achieved overall code size reductions. As can be seen, the code size overhead due to the decompressor varies between 2% (`gsm`, `gsm_enc` and `mpeg2`) up to 15% (`cjpeg_transupp`) only, compared to the benchmarks' original code size.

10.6 Conclusions

This article presented a survey of work done in the field of compiler techniques for real-time systems in the authors' group during the past 10 years. Origin of all these activities was the collaborative research project PREDATOR funded by the European 7th Framework Programme. During this project, seminal work was carried out in order to design predictable yet efficient embedded systems. A couple of scientific challenges has been identified that have initially been considered during PREDATOR and that, due to their complexity, required continuous research effort over many years even after the end of this collaborative research project. This article summarized these compiler-centric activities and their corresponding scientific challenges:

Challenge #1: Integration of task coordination into WCET-aware compilation
Challenge #2: Analysis and optimization of Multi-Processor Systems on Chip
Challenge #3: Predictable multi-objective compiler optimizations

Despite the advances in the field of compilation for real-time systems achieved in the past years, we expect that a continuation of this effort is necessary in the future. This is motivated by the trend towards massively parallel embedded real-time systems on the one hand, which still requires dedicated analyses and optimizations that are capable to support current and future many-core architectures. On the other hand, the simultaneous trade-off of various optimization objectives and the corresponding systematic exploration of the design space is still an unsolved problem for optimizing compilers. Last but not least, another important driver for future research is the increasing complexity of the involved system- and code-level analyses and optimizations which needs to be managed to obtain automated design tools that are usable in practice even for highly sophisticated and massively parallel systems.

Acknowledgments Parts of the work surveyed in this article received funding from Deutsche Forschungsgemeinschaft (DFG) under project No. 200265263 and 380772147. Other parts received funding from the European Union's 7th Framework Programme under grant agreement No. 216008 (PREDATOR) and from the Horizon 2020 research and innovation programme under grant agreement No. 779882 (TEAMPLAY).

References

1. AbsInt Angewandte Informatik GmbH, aiT: worst-case execution time analyzers (2020). http://www.absint.com/ait
2. K. Albers, F. Slomka, An event stream driven approximation for the analysis of real-time systems, in *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS)* (2004). https://doi.org/10.1109/EMRTS.2004.1311020

3. S.K. Baruah, Dynamic- and static-priority scheduling of recurring real-time tasks. Real-Time Syst. **24**, 93–128 (2003). https://doi.org/10.1023/A:1021711220939
4. E. Bini, G.C. Buttazzo, Measuring the performance of schedulability tests. Real-Time Syst. **30**, 129–154 (2005). https://doi.org/10.1007/s11241-005-0507-9
5. European Commission, Grant Agreement for FP7-ICT-216008 PREDATOR (2007)
6. European Commission, Design for predictability and efficiency (2017). https://cordis.europa.eu/project/rcn/85432
7. H. Falk, J.C. Kleinsorge, Optimal static WCET-aware scratchpad allocation of program code, in *Proceedings of the 46th Design Automation Conference (DAC)* (2009). https://doi.org/10.1145/1629911.1630101
8. H. Falk, P. Lokuciejewski, A compiler framework for the reduction of worst-case execution times. Real-Time Syst. **46**, 251–300 (2010). https://doi.org/10.1007/s11241-010-9101-x
9. H. Falk, S. Altmeyer, P. Hellinckx, et al., TACLeBench: a benchmark collection to support worst-case execution time research, in *Proceedings of the 16th International Workshop on Worst-Case Execution Time Analysis (WCET)* (2016). https://doi.org/10.4230/OASIcs.WCET.2016.2
10. K. Gresser, An event model for deadline verification of hard real-time systems, in *Proceedings of the 5th Euromicro Workshop on Real-Time Systems (ECRTS)* (1993). https://doi.org/10.1109/EMWRT.1993.639067
11. J. Gustafsson, A. Betts, A. Ermedahl, B. Lisper, The Mälardalen WCET benchmarks: past, present and future, in *Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis (WCET)* (2010). https://doi.org/10.4230/OASIcs.WCET.2010.136
12. R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, R. Ernst, System level performance analysis – the SymTA/S approach, in *IEE Proceedings – Computers and Digital Techniques* (2005). https://doi.org/10.1049/ip-cdt:20045088
13. A. Hidayat, FastLZ – free, open-source, portable real-time compression library (2007). http://fastlz.org
14. M. Jacobs, S. Hahn, S. Hack, WCET analysis for multi-core processors with shared buses and event-driven bus arbitration, in *Proceedings of the 23rd International Conference on Real-Time Networks and Systems (RTNS)* (2015). https://doi.org/10.1145/2834848.2834872
15. M. Joseph, P.K. Pandya, Finding response times in a real-time system. Comput. J. **29**, 390–395 (1986). https://doi.org/10.1093/comjnl/29.5.390
16. C. Lee, M. Potkonjak, W.H. Mangione-Smith, MediaBench: a tool for evaluating and synthesizing multimedia and communications systems, in *Proceedings of the 30th Annual International Symposium on Microarchitecture (1997)*. https://doi.org/10.1109/MICRO.1997.645830
17. Y.T.S. Li, S. Malik, Performance analysis of embedded software using implicit path enumeration, in *Proceedings of the Design Automation Conference (DAC)* (1995). https://doi.org/10.1145/217474.217570
18. C.L. Liu, J.W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM (1973). https://doi.org/10.1145/321738.321743
19. P. Lokuciejewski, H. Falk, P. Marwedel, WCET-driven, code-size critical procedure cloning, in *Proceedings of the 11th International Workshop on Software and Compilers for Embedded Systems (SCOPES), Munich* (2008), pp. 21–30
20. P. Lokuciejewski, S. Plazar, H. Falk, P. Marwedel, L. Thiele, Multi-objective exploration of compiler optimizations for real-time systems, in *Proceedings of the 13th International Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC)* (2010). https://doi.org/10.1109/ISORC.2010.15
21. P. Lokuciejewski, S. Plazar, H. Falk, P. Marwedel, L. Thiele, Approximating Pareto optimal compiler optimization sequences – a trade-off between WCET, ACET and code size. Softw. Pract. Exp. (2011). https://doi.org/10.1002/spe.1079

22. A. Luppold, H. Falk, Code optimization of periodic preemptive hard real-time multitasking systems, in *Proceedings of the 18th International Symposium on Real-Time Distributed Computing (ISORC)* (2015). https://doi.org/10.1109/ISORC.2015.8
23. A. Luppold, H. Falk, Schedulability aware WCET-optimization of periodic preemptive hard real-time multitasking systems, in *Proceedings of the 18th International Workshop on Software & Compilers for Embedded Systems (SCOPES)* (2015). https://doi.org/10.1145/2764967.2771930
24. A. Luppold, H. Falk, Schedulability-aware SPM allocation for preemptive hard real-time systems with arbitrary activation patterns, in *Proceedings of Design, Automation and Test in Europe (DATE)* (2017). https://doi.org/10.23919/DATE.2017.7927149
25. A. Luppold, D. Oehlert, H. Falk, Evaluating the performance of solvers for integer-linear programming. Tech. Rep., Hamburg University of Technology (2018). https://doi.org/10.15480/882.1839
26. K. Muts, A. Luppold, H. Falk, Compiler-based code compression for hard real-time systems, in *Proceedings of the 22nd International Workshop on Software and Compilers for Embedded Systems (SCOPES)* (2019). https://doi.org/10.1145/3323439.3323976
27. D. Oehlert, S. Saidi, H. Falk, Compiler-based extraction of event arrival functions for real-time systems analysis, in *Proceedings of the 30th Euromicro Conference on Real-Time Systems (ECRTS)* (2018). https://doi.org/10.4230/LIPIcs.ECRTS.2018.4
28. N. Piontek, Instruktionsscheduling für harte Multi-Core Echtzeitsysteme mit gemeinsam genutztem Datenbus. Masters Thesis, Hamburg University of Technology (TUHH) (2018)
29. S. Plazar, P. Lokuciejewski, P. Marwedel, WCET-aware software based cache partitioning for multi-task real-time systems, in *Proceedings of the 9th International Workshop on Worst-Case Execution Time Analysis (WCET)* (2009). https://doi.org/10.4230/OASIcs.WCET.2009.2286
30. A. Schranzhofer, R. Pellizzoni, J.J. Chen, L. Thiele, M. Caccamo, Worst-case response time analysis of resource access models in multi-core systems, in *Proceedings of the Design Automation Conference (DAC)* (2010). https://doi.org/10.1145/1837274.1837359
31. A. Schranzhofer, J.J. Chen, L. Thiele, Timing analysis for TDMA arbitration in resource sharing systems, in *Proceedings of 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)* (2010). https://doi.org/10.1109/RTAS.2010.24
32. S. Steinke, M. Knauer, L. Wehmeyer, P. Marwedel, An accurate and fine grain instruction-level energy model supporting software optimizations, in *Proceedings of the International Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS). Yverdon-Les-Bains* (2001)
33. V. Suhendra, T. Mitra, A. Roychoudhury, T. Chen, WCET centric data allocation to scratchpad memory, in *Proceedings of the 26th IEEE Real-time Systems Symposium (RTSS)* (2005). https://doi.org/10.1109/RTSS.2005.45
34. The PREDATOR Consortium, PREDATOR – design for predictability and efficiency (2011). https://www.predator-project.eu
35. L. Thiele, S. Chakraborty, M. Naedele, Real-time calculus for scheduling hard real-time systems, in *The 2000 IEEE International Symposium on Circuits and Systems. Proceedings. ISCAS 2000 Geneva*, vol. 4 (2000), pp. 101–104
36. J.S. Turner, New directions in communications (or which way to the information age?). IEEE Commun. Mag. (1986). https://doi.org/10.1109/MCOM.1986.1092946
37. UTDSP Benchmark Suite (2019). http://www.eecg.toronto.edu/$\sim$corinna/DSP/infrastructure/UTDSP.html

38. R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister, C. Ferdinand, Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. Trans. Comput.-Aid. Des. Integr. Circuits Syst. **28**, 966–978 (2009). https://doi.org/10.1109/TCAD.2009.2013287
39. J. Xu, A method for adjusting the periods of periodic processes to reduce the least common multiple of the period lengths in real-time embedded systems, in *Proceedings of the International Conference on Mechatronic and Embedded Systems and Applications (MESA)* (2010). https://doi.org/10.1109/MESA.2010.5552058