Efficient and Effective Multi-Objective Optimization for Real-Time Multi-Task Systems

Shashank Jadhav
Hamburg University of Technology, Germany

Heiko Falk
Hamburg University of Technology, Germany

Abstract

Embedded real-time multi-task systems must often not only comply with timing constraints but also need to meet energy requirements. However, optimizing energy consumption might lead to higher Worst-Case Execution Time (WCET), leading to an un-schedulable system, as frequently executed code can easily differ from timing-critical code. To handle such an impasse in this paper, we formulate a Metaheuristic Algorithm-based Multi-objective Optimization (MAMO) for multi-task real-time systems. But, performing multiple WCET, energy, and schedulability analyses to solve a MAMO poses a bottleneck concerning compilation times. Therefore, we propose two novel approaches – Path-based Constraint Approach (PCA) and Impact-based Constraint Approach (ICA) – to reduce the solution search space size and to cope with this problem. Evaluations showed that PCA and ICA reduced compilation times by 85.31% and 77.31%, on average, over MAMO. For all the task sets, out of all solutions found by ICA–FPA, on average, 88.89% were on the final Pareto front.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Software and its engineering → Compilers; Mathematics of computing → Discrete mathematics

Keywords and phrases Real-time systems, Multi-objective optimization, Metaheuristic algorithms, Compilers, Design space reduction

Digital Object Identifier 10.4230/OASIcs.WCET.2023.5

Funding This work is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 779882.

1 Introduction

Modern real-time embedded systems are subject to strict constraints and must meet functional and temporal requirements, such as execution time and energy consumption. Failure to meet such constraints might lead to disastrous consequences, e.g., airbag deployment systems. For multi-task systems, schedulability is an important criterion. This paper proposes multi-objective optimization for multi-task systems that simultaneously considers WCET, energy, and schedulability for multi-task systems. The proposed framework utilizes two metaheuristic algorithms, namely Strength Pareto Evolutionary Algorithm (SPEA) [26] and Flower Pollination Algorithm (FPA) [25], to solve a static Scratchpad Memory (SPM) allocation-based MAMO problem.

A program might have different WCET- and energy-critical paths. Moreover, minimizing WCET and energy of one particular task might negatively affect others, resulting in an un-schedulable system. Furthermore, the previous single-objective optimizations treated schedulability as a constraint, which could limit design space exploration. Therefore, the proposed framework considers schedulability as an objective, which enables the identification of a Pareto front containing completely- and partially-scheduled multi-task systems. Depending on hard- and soft-real-time requirements, in the end, the system designer could choose the needed Pareto-optimal solution.

The presented MAMO approach is iterative, requires objective evaluations at every iteration, and deals with three objectives, the analysis for which can be very time-consuming. To address this issue, we introduce two novel objective-dependent approaches that generate constraints at each iteration and reduce the solution space size and the total compilation time. The first approach, PCA, uses worst- and average-case execution path information to constrain the solution space, and the second approach, ICA, uses an impact metric to constrain and reduce the solution space size. Evaluations in this paper clearly show that worst- and average-case execution information can be utilized to improve MAMO speed and solution quality. Therefore, the key contributions of this paper are:

- We formulated and solved SPM allocation-based 3-dimensional MAMO problem.
- We proposed two approaches – PCA and ICA – to reduce the solution space size.
- PCA and ICA reduced compilation times by 85.31% and 77.31%, on average, over MAMO.
- On average, 88.89% solutions found by ICA–FPA were on the final Pareto front.

This paper is outlined as follows: Sec. (2) overviews the related work. Sec. (3) defines the MAMO problem for static SPM allocation. Sec. (4) introduces the MAMO framework. Sec. (5) and (6) propose PCA and ICA, respectively. Sec. (7) presents the evaluation results. Sec. (8) present conclusions and a future work discussion.

2 Related Work

In the past, many approaches focused on SPM allocation-based single-objective WCET- or energy-aware single-task optimizations [2, 7, 12, 14, 15, 22]. Moreover, some research has considered schedulability-aware single objective optimization for multi-task systems, where schedulability is treated as a constraint [16, 17]. In contrast, this paper proposes a compiler-level multi-objective optimization for multi-task systems which can simultaneously minimize WCET, energy consumption, and schedulability objectives.

Heuristic approaches like greedy algorithms may not be optimal for multi-objective optimizations, and ILP-based approaches are well-suited for single-objective optimization. Therefore, we use metaheuristic algorithms that employ problem-independent search strategies to solve the MAMO problem. Zitzler et al. [26] introduced the SPEA algorithm, where fitness assignment is done based on the co-evolution principle. FPA, inspired by the pollination process seen in flowering plants, is a Nature-Inspired metaheuristic Algorithm (NIA) introduced by Yang et al. [24]. SPEA outperforms other EAs such as NSGA, VEGA, etc. [27], and FPA performed better than VEGA, NSGA-II, MODE, etc. [25]. Therefore, in this paper, we chose to use SPEA and FPA to solve the MAMO problem.

Evaluating every solution iteratively using metaheuristics can pose a huge bottleneck. In such scenarios, machine learning techniques can be used to predict WCET and energy objectives [10, 21]. But, their accuracy varies on hyperparameter tuning, data sample size, and the underlying machine learning model. Previously, we proposed an approximation model to approximate WCET and energy consumption of a program, which was used to perform SPM allocation [13]. This approximation model relies on the underlying compiler optimization, i.e., SPM allocation, and is not generic to any compiler-level optimization. In this paper, we propose two approaches that rely on the objectives of MAMO, specifically WCET and energy and do not rely on the underlying compiler optimization.

3 Multi-Task Multi-Objective Problem

In this section, we propose a 3-dimensional MAMO problem for a multi-task system, where we treat schedulability as an objective to minimize rather than a constraint, which allows for better solution space exploration and maintains pressure on metaheuristic algorithms during solution selection. Treating schedulability as a constraint might lead to the rejection of solutions that violate this constraint and hinder proper solution space exploration. Moreover, as the schedulability objective depends on WCET, we could consider only schedulability as a minimization objective, but it could also hinder solution space exploration. For example, a multi-task system could become schedulable, and more schedulable solutions with lower WCET values could exist within the solution space. Lastly, WCET and energy consumption objectives can contradict each other. Therefore, we simultaneously consider WCET, energy consumption, and schedulability as objectives. The proposed MAMO problem for a multi-task system is mathematically formulated as follows:

$$\begin{aligned} \min_{x} \quad & F(x) = (F_1(x), F_2(x), F_3(x)) \\ \text{subject to} \quad & g(x) = \sum_{t=1}^{T}\sum_{v=1}^{p^t} B^{t_v} x^{t_v} - S_{SPM} - \sum_{t=1}^{T}\sum_{v=1}^{p^t-1} s^{t_v} |x^{t_v} - x^{t_v+1}| \leq 0 \end{aligned} \tag{1}$$

where $x = (x^1, .., x^T) \in \{0,1\}^d$ is a d-dimensional binary decision vector for a multi-task set Γ. x^t is a binary decision vector for a single task $\tau^t \in \Gamma$, where $t = \overline{1,T}$, and T is the total number of tasks. SPM allocation is a compiler optimization, where we move Basic Blocks (BB) from slow Flash to SPM. The decision of placing a BB in SPM or Flash is realized by an element $x^{t_v} \in \{0,1\}$ of the decision vector, where $v = \overline{1,p^t}$, p^t is the total number of BBs in the t^{th} task, and $d = \sum_{t=1}^{T}\sum_{v=1}^{p^t} v$ is the total number of BBs in the multi-task system.

$g(x) \leq 0$ is the SPM size constraint condition, where B^{t_v} is the code size of BB^{t_v}, BB^{t_v} is the v^{th} BB in the t^{th} task, and S_{SPM} represents the SPM size. The term $|x^{t_v} - x^{t_v+1}|$ determines if extra jump correction cost is needed, i.e., if t_v^{th} BB and the succeeding $(t_v+1)^{\text{th}}$ BB are in different memories, then we need to perform jump correction and add the extra jump correction cost [18]. Furthermore, s^{t_v} is an architecture-dependent term representing the jump correction code size. For ARM7TDMI architecture, s^{t_v} is modeled as follows: $s^{t_v} = 16$ if the basic block BB^{t_v} ends with a jump instruction, and in case of calls, conditional jumps, or fall-though instructions $s^{t_v} = 16$. Extra spill code is added if a free register is not available, which increments the jump correction cost, i.e., $s^{t_v} += 4$.

$F(x) = (F_1(x), F_2(x), F_3(x))$ is the 3-dimensional objective function. $F_1(x) = \sum_{t=1}^{T} W^t$ and $F_2(x) = \sum_{t=1}^{T} E^t$ are the total WCET and energy values for the binary decision vector x, where W^t and E^t are WCET and energy consumption of task $\tau^t \in \Gamma$, respectively. $F_3(x) = \sum_{t=1}^{T} \rho_t W^t$ is the schedulability objective. $\rho \in \{0,1\}^T$ is a T-dimensional binary vector, where $1's$ indicate that the task τ^t is a task removed from Γ, i.e., $\tau^t \in \Gamma_r$, where Γ_r denotes a set of tasks removed from Γ, such that the system of remaining tasks is schedulable. We use the ILP-based schedulability analyzer [16] to calculate the number of tasks needed to be removed from the system to achieve schedulability. Accordingly, if a system with T tasks is schedulable and all the tasks safely meet their deadlines, the analyzer will return 0. Contrary, in a worst-case scenario, the analysis will return T. MAMO returns a set of Pareto-optimal solutions that can be fully or partially schedulable and indifferent in terms of WCET and energy, which allows the system designer to choose a suitable solution from the set of Pareto-optimal binaries based on the requirements during runtime.

Algorithm 1 SPM allocation-based MAMO.

1: **Initialization:** Initialize the initial population, perform jump corrections, and evaluate them.
2: **Input:** Initialized population and stopping criteria
3: **Output:** Pareto-optimal solution set
4: **while** stopping criteria is not fulfilled **do** ▷ Iterate over all generations
5: **for** $j = 1 : N$ **do** ▷ Iterate over all individuals
6: Update the individual using update operators
7: Repair the individual if needed and perform jump correction
8: Evaluate the individual
9: Using the selection operator, update to next generation

4 Multi-objective optimization

To solve the compiler-level MAMO problem for multi-task systems, we use the WCET-aware C Compiler (WCC) [6] framework. We solve the MAMO problem using a metaheuristic algorithm and try to minimize WCET, energy consumption, and schedulability. Let $X \subset \{0,1\}^d$ be the search space of the MAMO defined in Sec. (3). $P_i \subset X$ is the population set with N individuals at generation $i = \overline{1, M}$, where M is the maximum number of generations. $x_{i,j} \in \{0,1\}^d$ is a d-dimensional binary individual vector at generation i, where $j = \overline{1, N}$.

Algorithm (1) presents the SPM allocation-based MAMO. The problem formulation presented in Sec. (3) is used to initialize MAMO (Line 1). After initialization, we solve MAMO by calling the metaheuristic algorithm (Lines 4-9). The metaheuristic algorithm uses an update operator to update an individual (Line 6). SPEA uses mutation and crossover [26], whereas FPA uses global and local pollination operators to update an individual [25]. The current population generation is updated after evaluations to the next using a selection operator. FPA and SPEA provide scalar fitness values to each individual and use Pareto Dominance [4] to update to the next generation. This paper considers two stopping criteria, the maximum number of generations and the maximum number of generations for which the population remains the same. After fulfilling the stopping criteria, the algorithm outputs the final Pareto-optimal solution set. The Algorithm (1) considers all the BBs in the multi-task system. The number of BBs defines the dimension of the solution space, which influences the second stopping criterion. Therefore, the smaller the solution space size, the quicker we might reach the second stopping criteria, leading to fewer individuals to evaluate. Therefore, we propose two novel approaches to reduce the solution space size in the following sections.

5 Path-based Constraint Approach

During optimization, exploring the whole solution space would be the most reliable way to find the Pareto-front. But, with limited time to perform optimization, we can consider objective-specific details to decrease the problem size. The WCET and energy objectives rely

Algorithm 2 Path-based Constraint Approach.

1: **Input:** Evaluated individual $x_{i,j}$
2: **Output:** Constraints for next individual $x_{i+1,j}$
3: Get $\mathcal{W}_{i,j}$ and $\mathcal{A}_{i,j}$, and Create $\mathcal{U}_{i,j}$.
4: **for** $t = 1 : T$ **do** ▷ Iterate over all tasks
5: **for** $v = 1 : p^t$ **do** ▷ Iterate over all basic blocks
6: **if** $BB_{i,j}^{t_v} \notin \mathcal{U}_{i,j}$ **then**
7: $x_{i+1,j}^{t_v} = 0$

Algorithm 3 Impact-based Constraint Approach.

1: **Input:** Evaluated individual $x_{i,j}$
2: **Output:** Constraints for next individual $x_{i+1,j}$
3: Create an empty set : $\mathcal{H}_{i,j}$ and an empty list of BBs : $\mathcal{B}_{i,j}$
4: **for** $t = 1 : T$ **do** ▷ Iterate over all tasks
5: **for** $v = 1 : p^t$ **do** ▷ Iterate over all basic blocks
6: $\mathcal{H}_{i,j} \longleftarrow (BB^{t_v}_{i,j}, \mathcal{M}^{t_v})$
7: **Sort** $(\mathcal{H}_{i,j})$ in the descending order of $\mathcal{M}^{t_v}$ values
8: **for** $q = 1 : d$ **do** ▷ Iterate over $(\mathcal{H}_{i,j})$
9: **if** $\sum_{b=1}^{\eta} B_b <= \alpha * S_{SPM}$ **then**
10: $\mathcal{B} \longleftarrow \mathcal{H}^{q,1}_{i,j}$
11: **for** $t = 1 : T$ **do** ▷ Iterate over all tasks
12: **for** $v = 1 : p^t$ **do** ▷ Iterate over all basic blocks
13: **if** $BB^{t_v}_{i,j} \notin \mathcal{B}_{i,j}$ **then**
14: $x^{t_v}_{i+1,j} = 0$

on the Worst-Case Execution Path (WCEP) and Average-Case Execution Path (ACEP) of a program, respectively [1, 23]. WCEP and ACEP are defined as the execution paths through a task's control flow graph that leads to their WCET and Average-Case Execution Time (ACET), respectively. Therefore, instead of exploring the entire search space, we constrain the solution space at each iteration using WCEP and ACEP information. As the WCEP and ACEP of a task set could differ, PCA considers the BBs on both paths. Let $\mathcal{W}_{i,j}$ and $\mathcal{A}_{i,j}$ be the set of BBs on WCEP and ACEP of the j^{th} individual of the i^{th} generation. For every individual, BBs on WCEP and ACEP can vary. Furthermore, let $\mathcal{U}_{i,j} := \mathcal{W}_{i,j} \bigcup \mathcal{A}_{i,j}$. Therefore, MAMO defined by Eq. (1) is extended by adding the following constraint.

$$x^{t_v}_{i+1,j} = 0, \quad \text{if } BB^{t_v}_{i,j} \notin \mathcal{U}_{i,j} \quad \forall t, v \tag{2}$$

Algorithm (2) describes the PCA approach proposed in this paper. PCA takes an evaluated individual $x_{i,j}$ as an input and provides constraints for the individual $x_{i+1,j}$ from the next generation (Lines 1-2). The sets of BBs, $\mathcal{W}_{i,j}$, $\mathcal{A}_{i,j}$, and $\mathcal{U}_{i,j}$, are created (line 3). If a BB is not on WCEP or ACEP, then that element of the individual vector for the next generation is constrained to 0, and the BB is placed in Flash, i.e., $x^{t_v}_{i+1,j} = 0$ (lines 4-7). This constraint is enforced by recombination and mutation operators for SPEA and local and global pollination operators for FPA. Therefore, the size and shape of the solution space can change and be differently constrained throughout the optimization run.

6 Impact-based Constraint Approach

PCA considered all BBs on the WCEP and ACEP to reduce the dimension of the solution space, but MAMO has an SPM size constraint, and SPMs are small in size. Consequently, we can assume that many BBs will not be assigned to SPM for large multi-task systems. Therefore, we propose the ICA Approach, which guides the optimization using the SPM size constraint, Worst-Case Execution Count (WCEC), and Average-Case Execution Count (ACEC) to constrain the solution space. WCEC and ACEC are defined as the number of times each basic block is executed in a worst- and average-case scenario, respectively. As the schedulability objective operates at the task level, we do not consider it for solution space reduction. While initializing ICA, we calculate the impact of each BB on the total WCET and energy when SPM allocation is not performed. We consider an individual F^* where all

the BBs are in the Flash and evaluate it by performing WCET and energy analyses. Let W_{F^*} and E_{F^*}, and $W^{t_v}_{F*}$ and $E^{t_v}_{F^*}$ be the total WCET and total energy, and the WCET and energy of $BB^{t_v}_{F^*}$, respectively. $BB^{t_v}_{F^*}$ is the v^{th} basic block of the t^{th} task of the individual F^*. Furthermore, we calculate the impact of each BB by using $\mathbf{W}^{t_v}_{F^*} := \frac{W^{t_v}_{F^*}}{W_{F^*}}$ and $\mathbf{E}^{t_v}_{F^*} := \frac{E^{t_v}_{F^*}}{E_{F^*}}$. Let $\mathcal{W}_{F^*}$ and $\mathcal{E}_{F^*}$ be the set of $\mathbf{W}^{t_v}_{F^*}$ and $\mathbf{E}^{t_v}_{F^*}$ $\forall t, v$, respectively for the individual F^*. After calculating impact values for each BB for the case where all BBs are in Flash, we call Algorithm (1) to solve the MAMO problem.

Algorithm (3) describes the proposed ICA approach used to reduce the solution space size. ICA takes an evaluated individual $x_{i,j}$ as an input and provides constraints for the individual $x_{i+1,j}$ from the next generation (Lines 1-2). ICA uses the following *impact* metric to constrain the solution space.

$$\mathcal{M}^{t_v} = \zeta \mathbf{W}^{t_v}_{F^*} * w^{t_v}_{i,j} + \beta \mathbf{E}^{t_v}_{F^*} * a^{t_v}_{i,j} \text{ ,where } \mathbf{W}^{t_v}_{F^*} \in \mathcal{W}_{F^*} \ \& \ \mathbf{E}^{t_v}_{F^*} \in \mathcal{E}_{F^*} \tag{3}$$

The terms $w^{t_v}_{i,j}$ and $a^{t_v}_{i,j}$ represent the WCEC and ACEC of $BB^{t_v}_{i,j}$ of the individual j at generation i, respectively. Furthermore, $\zeta, \beta \in [0, 1]$ and $\zeta + \beta = 1$ are the positive constant weights assigned to WCET and energy terms. We can use ζ and β to adjust the weight of the objectives. During SPM allocation, the BBs on WCEP and ACEP affect the program's total WCET and energy. But, BBs having higher WCEC and ACEC values does not indicate that the BB will have higher WCET and energy values. Therefore, the WCEC and ACEC term in Eq. (3) is multiplied by the WCET and energy impact terms to calculate the BB's impact. The bigger the value of $\mathcal{M}^{t_v}$, the higher the overall impact of $BB^{t_v}_{i,j}$ on the total WCET and energy of the individual.

An empty set $\mathcal{H}_{i,j}$ of the ordered pairs of BB and *impact* metric is initialized (line 3), i.e., $(BB^{t_v}_{i,j}, \mathcal{M}^{t_v}) \in \mathcal{H}_{i,j}$ $\forall t, v$. Furthermore, an empty list $\mathcal{B}_{i,j}$, which will contain the list of BBs selected by this heuristic, is initialized (line 3). We iterate through the multi-task system, i.e., $d = \sum_{t=1}^{T} \sum_{v=1}^{p^t} v$ times, to fill the set $\mathcal{H}_{i,j}$ and sort this set in the descending order of $\mathcal{M}^{t_v}$ values (lines 4-7). For selecting the BBs, we iterate through $\mathcal{H}_{i,j}$ and add BBs to $\mathcal{B}_{i,j}$ until the total size of the BBs selected, i.e., $\sum_{b=1}^{\eta} B_b$, is not greater than $\alpha * S_{SPM}$, where α is a positive constant, and S_{SPM} is the SPM size (Lines 8-10). Furthermore, B_b is the size of the b^{th} BB in $\mathcal{B}_{i,j}$ and $\eta \in \mathbb{N}$ such that $1 \leq \eta \leq d$ represents the total number of BBs which are not constrained for the $(i+1)^{\text{th}}$ generation. Using this heuristic, we select the BBs that have the highest impact on the WCET and energy of the code. We further constrain the solution space by limiting the number of the BBs selected by $\alpha * S_{SPM}$. If a BB is not in $\mathcal{B}_{i,j}$, then that element of the individual for the next generation is constrained to 0, i.e., $x^{t_v}_{i+1,j} = 0$ (Lines 11-14). This constraint is enforced by SPEA's recombination and mutation operators and FPA's local and global pollination operators to update the individual to $(i+1)^{\text{th}}$ generation. Therefore, if the BB is not in $\mathcal{B}_{i,j}$, i.e., $BB^{t_v}_{i,j} \notin \mathcal{B}_{i,j}$, then $x^{t_v}_{i+1,j}$ BB of individual j will be placed in Flash during the $(i+1)^{\text{th}}$ generation. Using this heuristic, we drastically constrain the solution space of big multi-task problems and perform MAMO within a limited timeframe. As ICA considers WCEC and ACEC information, it relies indirectly on the properties considered in PCA. But, unlike PCA, the ICA does not ignore the BBs not on WCEP and ACEP, but they have a low priority.

7 Evaluation

In this section, the evaluations compare the MAMO approach with PCA and ICA. The evaluations include compilation times, Pareto fronts, and quality indicators comparison for task sets consisting of 2, 4, 6, and 8 tasks. For each, 10 task sets are randomly generated

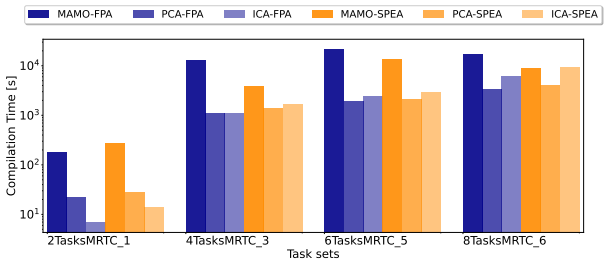

Figure 1 Compilation times required to perform MAMO, PCA, and ICA.

and are un-schedulable by construction. Multiple single-task benchmarks from the MRTC suite [9], with loop bound annotations from the TACLeBench [5], were combined into a multi-tasking task set. Optimization flag `-O2`, which enables several ACET-oriented compiler optimizations, was used, and SPM size was set to 60% of the code size of each task set to increase the pressure on MAMO. A timeout value of 200 h is set for the optimization. If the optimization does not finish within this time limit, the final Pareto front is generated from the last generation, and the final results are output. The SPM allocation code generated by WCC is for ARM7TDMI architecture. For the sake of brevity, the figures show results for four randomly chosen task sets[1]. WCC uses an external WCET analyzer called *aiT* [1] for WCET analysis and an internal schedulability analyzer [16] for schedulability analysis. Furthermore, WCC uses a cycle-true instruction set simulator from Synopsys called Virtualizer [11] to get ACEC and ACET data. The energy analyzer within WCC uses the energy model proposed by Roth et al. [20] and average-case data to perform the analysis [23].

For the FPA-based optimization, the switch probability between the local and global pollination is $p_s = 0.8$. According to [25, 19], for FPA, the positive integer $\lambda = 1.5$ for the standard gamma function, and the scaling factor $\gamma = 0.1$ works well. For SPEA-based optimization, the external population set size is 10. The crossover probability is 0.8, and the mutation probability is 0.2. The population size is 10, the first stopping criterion –the maximum number of generations– is 80, and the second stopping criterion –the maximum number of generations for which the population remains the same– is 10. PCA does not need any extra parameter settings. But, for ICA, we set the weights for $\mathcal{M}$ as $\zeta = 0.5$ and $\beta = 0.5$ so that the WCET and energy objectives are equally weighted while selecting the BBs. Furthermore, $\alpha = 0.9$, i.e., we allow the BBs 0.9 times the SPM size during selection. α is less than 1 to accommodate the extra code inserted during jump correction. The results from these evaluations are valid for the algorithm parameters described above.

7.1 Compilation Times

Fig. (1) compares the compilation times for MAMO, PCA, and ICA on four task sets. The x- and y-axis represent task sets and compilation times, respectively. Each task set has six bars – the first three are the results for FPA, and the last three for SPEA, respectively, for MAMO, PCA, and ICA. The figure shows that MAMO took more time to output the final Pareto front than PCA and ICA. Overall evaluations show that PCA and ICA achieved 85.31% and 77.31% overall reduction in compilation times, respectively, compared to MAMO. Moreover, PCA–FPA achieved the most reduction in compilation time. The timeout value is hit by

[1] All the remaining figures can be made available at the readers' request.

17, 6, and 7 task sets in the case of MAMO, PCA, and ICA, respectively. MAMO–FPA, on average, required 184.41% higher compilation time than MAMO–SPEA. PCA–FPA and ICA–FPA required 26.8% and 46.99% less compilation times than PCA–SPEA and ICA–SPEA, respectively. The improved performance of FPA over SPEA in PCA and ICA may be due to constraints on the search space, which make FPA update strategies for PCA and ICA more effective in converging to the Pareto-optimal front.

The overall decrease in compilation times achieved using PCA and ICA is due to the search space reduction and the second stopping criterion. The total number of BBs in the task set determines the search space size. To calculate the reduction in the search space for PCA, we first calculated the total number of BBs not constrained by Eq. (2). As the number of BBs on WCEP and ACEP may vary, we calculated the average number of BBs on WCEP and ACEP over all populations. Similarly, for ICA, the number of BBs not constrained according to the Algorithm (3) are calculated and averaged over all the populations. PCA and ICA, on average, achieved 60.06% and 87.638% reduction in the search space, respectively. The reduction in the search space can cause the optimization to hit the second stopping criteria faster. But, the average reduction in the search space cannot directly reflect the decrease in compilation times. If the metaheuristics find better solutions in every generation, the second stopping criteria is not fulfilled. E.g., even though ICA achieved a higher reduction in the search space size, PCA–SPEA achieved the most reduction in compilation time on average.

7.2 Pareto Fronts

Obtaining a true Pareto front to MAMO problem described in Sec. (3) is ambitious. Under a realistic assumption that the true Pareto front is unknown, we define a new set $\mathcal{P}$ that represents its approximation. Let A and B be two Pareto fronts returned by FPA and SPEA, respectively. The new set $\mathcal{P}$ is defined as the set of all non-dominated points of the union of the sets A and B, i.e., $\mathcal{P} = \{p_i | \forall p_i \nexists c_j \in (A \cup B) \prec p_i\}$. For each task set, we obtain the final Pareto front ($\mathcal{P}$) from the union of all the Pareto optimal solutions obtained from all the approaches. Fig. (2) shows the solutions for four task sets found by MAMO, PCA, ICA, and the standard `-O2` optimization level in the form of Pareto fronts. We represent WCET, energy, and schedulability values on the $x-$, $y-$, and $z-$axis, respectively. In the legend of the sub-figure, we show the number of solutions returned by each approach. Furthermore, we show the number of solutions on $\mathcal{P}$ out of the total solutions. To distinguish the solutions on $\mathcal{P}$, they are highlighted using numbers.

For the task set 2TasksMRTC_1, we see that all solutions by MAMO and ICA are on $\mathcal{P}$. One MAMO–FPA solution (Solution 2) is scheduled, whereas the other is schedulable if one task is removed from the task set. Similarly, one MAMO–SPEA solution out of three (Solution 5) has a schedulable task set. ICA–FPA and ICA–SPEA found one completely scheduled solution each (Solution 6 and 7) on $\mathcal{P}$, converging on the same solution. None of the solutions obtained by PCA are on $\mathcal{P}$. Considering the compilation times by ICA and MAMO, we can conclude that ICA performed better for this task set. Besides, when we compare the solutions on $\mathcal{P}$ with the `-O2` solution, we see on average 34.02%, 78.69%, and 91.781% decrease in WCET, energy consumption, and schedulability objectives, respectively. The 91.781% average decrease in the schedulability objective indicates that $\mathcal{P}$ consists of partially scheduled solutions too.

For task set 4TasksMRTC_3, ICA–FPA clearly outperforms others. Both ICA–FPA's solutions on $\mathcal{P}$ are completely schedulable. For the task set 6TasksMRTC_5, PCA–FPA and ICA–FPA found three and two solutions on $\mathcal{P}$, respectively. ICA–FPA solutions are

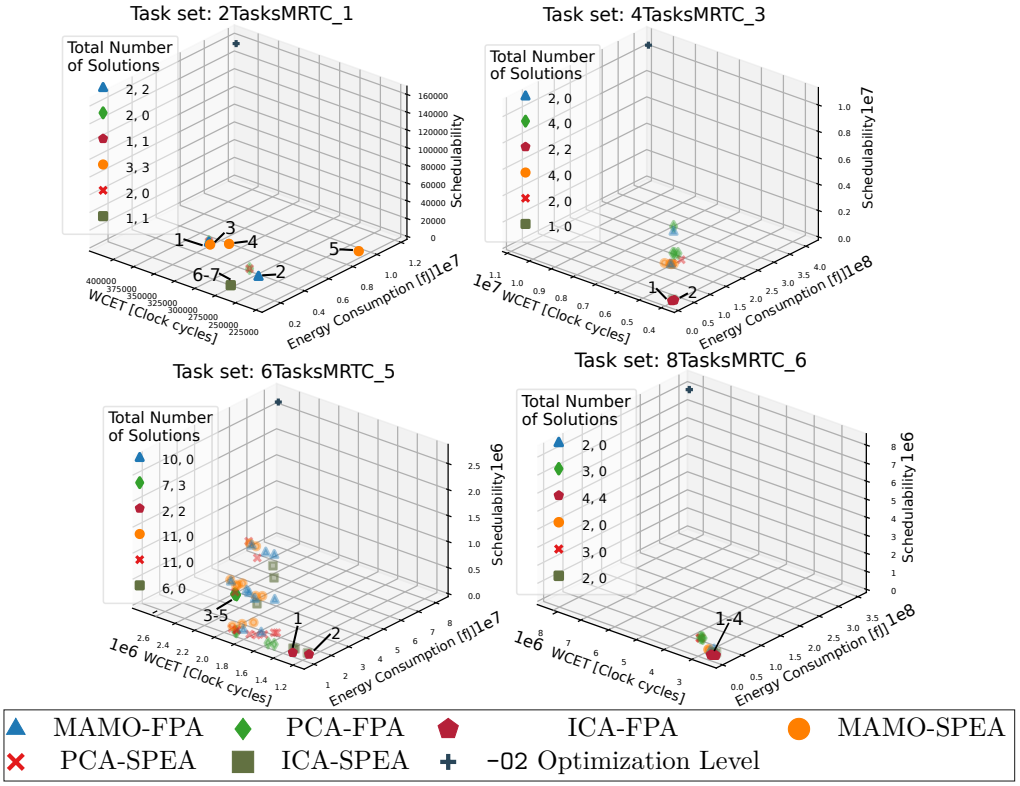

Figure 2 Solutions Obtained by MAMO, PCA, ICA, and `-O2` runs for multi-task system.

completely schedulable, and PCA–FPA solutions are schedulable if one task is removed from the task set. But, PCA–FPA found these solutions faster than ICA–FPA (cf. Fig. (1)). For task set 8TasksMRTC_6, ICA–FPA was able to find better solutions overall. We can see that the time taken by PCA–FPA is lower than ICA–FPA (cf. Fig. (1)). But, the quality of the solutions obtained by ICA is much better than PCA. Consequently, we can say that ICA–FPA performs better for this task set. Furthermore, when we compare the solutions on $\mathcal{P}$ with the `-O2` solution, we see on average 68.23%, 97.09%, and 100% decrease in WCET, energy consumption, and schedulability objectives, respectively. The 100% decrease in the schedulability objective indicates that all the solutions on $\mathcal{P}$ are completely schedulable.

From overall evaluations, for all the task sets, out of all solutions found by MAMO for FPA and SPEA, on average, 10.24% and 13.39% of solutions were on $\mathcal{P}$. PCA for FPA and SPEA had, on average, 11.27% and 5.93% of solutions on $\mathcal{P}$. Finally, ICA for FPA and SPEA algorithms had 88.89% and 4.81% of solutions on $\mathcal{P}$, respectively, on average. Therefore, we can say that ICA–FPA provided most solutions on $\mathcal{P}$.

7.3 Quality Metrics

We use the following quality metrics to evaluate and compare the quality of multi-objective optimization approaches. *Coverage* (C_A) [26] is a quality metric that describes the total number of dominated points in a set A. The lower the value of C_A, the better. The *Non-Dominance Ratio* (NDR_A) [8] is another quality metric measuring the ratio of non-dominated solutions contributed by a particular solution set A to the non-dominated solutions provided by all solution sets. The higher the value of the non-dominance ratio, the better. The

Non-Dominated Solutions (NDS_A) [3] is the last considered quality metric that calculates the number of non-dominated solutions concerning the set A itself, when compared to $\mathcal{P}$. The higher the value of non-dominated solutions, the better.

Table 1 Comparing FPA & SPEA.

Quality metric	Number of Task Sets					
	MAMO		*PCA*		*ICA*	
	FPA	SPEA	FPA	SPEA	FPA	SPEA
C_A	21	19	23	17	37(3)	0(3)
NDR_A	19(2)	19(2)	20(3)	17(3)	40	0
NDS_A	21	19	23	17	37(3)	0(3)

Table 2 Comparing MAMO, PCA, & ICA.

Quality metric	Number of Task Sets					
	FPA			*SPEA*		
	MAMO	PCA	ICA	MAMO	PCA	ICA
C_A	3(4)	2(4)	27(3)	3(3)	1(3)	0(3)
NDR_A	3(4)	2(4)	27(3)	3(3)	1(3)	0(3)
NDS_A	3(4)	2(4)	27(3)	3(3)	1(3)	0(3)

We compared quality metrics to evaluate the quality of the obtained solutions. We first compared solutions obtained using FPA and SPEA algorithms for each approach. For this comparison, $\mathcal{P}$ is generated individually for MAMO, PCA, and ICA by combining the final Pareto fronts of their respective FPA and SPEA runs. Table (1) provides the total number of task sets for which FPA and SPEA performed better for each approach. During evaluations, we encountered task sets for which the quality metrics were indifferent to each other. The total number of such indifferent task sets is indicated using brackets within the table. From an overall comparison, we can say that SPEA and FPA algorithms provided relatively the same quality of solutions for MAMO and PCA, and FPA performed outright better for ICA.

Furthermore, we compared MAMO, PCA, and ICA approaches in terms of the above-mentioned quality metrics. For this comparison, $\mathcal{P}$ is generated by combining the final Pareto fronts of all the approaches. Table (2) provides the total number of task sets for which MAMO, PCA, and ICA performed better. The total number of task sets with indifferent quality metrics are indicated within the table using brackets. From an overall comparison, we can clearly see that ICA–FPA provided better quality solutions for most task sets. Although it is difficult to know beforehand which approach will deliver the better Pareto-optimal solutions due to the non-deterministic nature of algorithms, we could find better results by constraining the solution space and focusing the optimization direction on the BBs that highly impact the objectives.

8 Conclusions

In this paper, we formulated a 3-dimensional SPM allocation-based MAMO and solved it using FPA and SPEA algorithms. As the compilation times required for the optimization can increase with the problem size, we introduced two new approaches, PCA and ICA, to cope with the MAMO problem size. All these approaches were able to find the trade-offs between schedulability, WCET, and energy consumption. Moreover, we compared the results obtained using FPA and SPEA for all three approaches. From evaluations, we, on average, achieved 85.31% and 77.31% reduction in compilation times using PCA and ICA compared to MAMO, respectively. Moreover, ICA–FPA found good quality solutions for 27 task sets and, on average, found 88.89% of the solutions on $\mathcal{P}$, which is the highest compared to other considered optimization approaches. Therefore, in this paper, we were able to show that clever integration of worst-case and average-case information within the optimization can lead to a drastic reduction in compilation times and help find better-quality solutions.

The two approaches discussed in this paper to reduce the solution space effectively reduced the runtime of the optimization and provided quality solutions. But, there were still some larger task sets that ran into timeouts during our evaluations. In the future, a

promising approach to further reduce the compilation times could be to partially replace time-consuming WCET and energy analyses with pessimistic WCET and energy estimations. In this paper, the approximated Pareto-optimal solution set returned by the optimization could consist of either fully or partially schedulable solutions. But a multi-task system could consist of tasks with both hard and soft timing constraints. Therefore, in the future, we could extend the proposed multi-objective formulation in which the system designer specifies tasks with hard and soft timing constraints. Based on these specifications, the compiler could either treat a task as part of the schedulability constraint or part of the schedulability objective, which could further constrain the solution space and provide the system designer more control over the optimization parameters and the compiler output. Furthermore, SPM allocation is just one optimization that we have considered as a multi-task multi-objective problem. Other compiler-based optimizations also have great potential, which might need objective-independent ways to constrain the solutions space. Therefore, in the future, we can consider hybrid algorithms, which combine relaxed ILPs and metaheuristic algorithms together to constrain the solution space in a problem-independent manner.

References

1 AbsInt Angewandte Informatik, GmbH. aiT Worst-Case Execution Time Analyzers, 2021.

2 Anuradha Balasundaram and Vivekanandan Chenniappan. Optimal code layout for reducing energy consumption in embedded systems. In *2015 International Conference on Soft-Computing and Networks Security (ICSNS)*, pages 1–5. IEEE, 2015.

3 Sanghamitra Bandyopadhyay and Arpan Mukherjee. An algorithm for many-objective optimization with reduced objective computations: A study in differential evolution. *IEEE Transactions on Evolutionary Computation*, 19(3):400–413, 2014.

4 Michael TM Emmerich and André H Deutz. A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Natural computing*, 17:585–609, 2018.

5 Heiko Falk, Sebastian Altmeyer, Peter Hellinckx, Björn Lisper, Wolfgang Puffitsch, Christine Rochange, Martin Schoeberl, Rasmus Bo Sørensen, Peter Wägemann, and Simon Wegener. Taclebench: A benchmark collection to support worst-case execution time research. In *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2016.

6 Heiko Falk and Paul Lokuciejewski. A Compiler Framework for the Reduction of Worst-Case Execution Times. *Real-Time Systems*, 46(2):251–298, 2010.

7 Heiko Falk, Sascha Plazar, and Henrik Theiling. Compile-Time Decided Instruction Cache Locking Using Worst-Case Execution Paths. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 143–148, 2007. DOI 10.1145/1289816.1289853.

8 Chi-Keong Goh and Kay Chen Tan. A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 13(1):103–127, 2008.

9 Jan Gustafsson, Adam Betts, Andreas Ermedahl, and Björn Lisper. The mälardalen wcet benchmarks: Past, present and future. In *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

10 Thomas Huybrechts, Siegfried Mercelis, and Peter Hellinckx. A new hybrid approach on wcet analysis for real-time systems using machine learning. In *18th International Workshop on Worst-Case Execution Time Analysis (WCET 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

11 Synopsys Inc. Comet system engineering ide, Online. URL: `http://www.synopsys.com`.

12 Yuriko Ishitobi, Tohru Ishihara, and Hiroto Yasuura. Code placement for reducing the energy consumption of embedded processors with scratchpad and cache memories. In *2007 IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia*, pages 13–18. IEEE, 2007.

13 Shashank Jadhav and Heiko Falk. Approximating wcet and energy consumption for fast multi-objective memory allocation. In *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, pages 162–172, 2022.

14 Andhi Janapsatya, Aleksandar Ignjatović, and Sri Parameswaran. A novel instruction scratchpad memory optimization method based on concomitance metric. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 612–617, 2006.

15 Yooseong Kim, David Broman, and Aviral Shrivastava. WCET-Aware Function-Level Dynamic Code Management on Scratchpad Memory. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(4):1–26, 2017.

16 Arno Luppold and Heiko Falk. Schedulability aware wcet-optimization of periodic preemptive hard real-time multitasking systems. In *Proceedings of the 18th International Workshop on Software and Compilers for Embedded Systems*, pages 101–104, 2015.

17 Arno Luppold and Heiko Falk. Schedulability-aware spm allocation for preemptive hard real-time systems with arbitrary activation patterns. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1074–1079. IEEE, 2017.

18 Dominic Oehlert, Arno Luppold, and Heiko Falk. Practical challenges of ilp-based spm allocation optimizations. In *Proceedings of the 19th International Workshop on Software and Compilers for Embedded Systems*, pages 86–89. ACM, 2016.

19 Douglas Rodrigues, Xin-She Yang, André Nunes De Souza, and João Paulo Papa. Binary flower pollination algorithm and its application to feature selection. In *Recent advances in swarm intelligence and evolutionary computation*, pages 85–100. Springer, 2015.

20 Mikko Roth, Arno Luppold, and Heiko Falk. Measuring and modeling energy consumption of embedded systems for optimizing compilers. In *Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems*, pages 86–89. ACM, 2018.

21 Akash Sachan and Bibhas Ghoshal. Learning based compilation of embedded applications targeting minimal energy consumption. *Journal of Systems Architecture*, 116:102116, 2021.

22 Stefan Steinke, Lars Wehmeyer, Bo-Sik Lee, and Peter Marwedel. Assigning program and data objects to scratchpad for energy reduction. In *Design, Automation & Test in Europe (DATE)*, pages 409–415, 2002.

23 TeamPlay Consortium. Deliverable D3.1 - Report on the TeamPlay Basic Compiler Infrastructure - Version 1.0, 2018.

24 Xin-She Yang. Flower pollination algorithm for global optimization. In *International conference on unconventional computing and natural computation*, pages 240–249. Springer, 2012.

25 Xin-She Yang, Mehmet Karamanoglu, and Xingshi He. Flower pollination algorithm: a novel approach for multiobjective optimization. *Engineering Optimization*, 46(9):1222–1237, 2014.

26 Eckart Zitzler. *Evolutionary algorithms for multiobjective optimization: Methods and applications*, volume 63. Citeseer, 1999.

27 Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.