Approximating WCET and Energy Consumption for Fast Multi-Objective Memory Allocation

Shashank Jadhav
shashank.jadhav@tuhh.de
Institute of Embedded Systems, TUHH
Hamburg, Germany

Heiko Falk
heiko.falk@tuhh.de
Institute of Embedded Systems, TUHH
Hamburg, Germany

ABSTRACT

Worst-Case Execution Time (WCET) is the most important design criterion in the domain of hard real-time systems. Most embedded systems also need to satisfy additional design criteria like, e.g., energy consumption. Performing WCET and energy analyses statically at compile-time can be time-consuming. Consequently, minimizing WCET and energy consumption of the code at the compiler level using multi-objective optimization can be a time-consuming process. In this paper, we propose an approximation model to quickly approximate the WCET and energy consumption of the code at compile-time. Instead of using traditional WCET and energy analyses, we exploit this approximation model to perform ScratchPad Memory (SPM) allocation-based multi-objective optimization. Furthermore, we solve the multi-objective optimization problem using metaheuristic algorithms and explore the trade-offs between WCET and energy consumption. Using the proposed approximation model, we achieved, on average, a 94.12% reduction in compilation time and maintained the quality of the Pareto optimal solutions while performing the multi-objective optimization. Furthermore, the approximation error while using the proposed approximation model was in an acceptable range of 2% - 4% on average.

CCS CONCEPTS

• **Computer systems organization** → *Embedded systems*; • **Software and its engineering** → *Compilers*; • **Computing methodologies** → *Modeling methodologies*; • **Mathematics of computing** → *Discrete optimization.*

KEYWORDS

Multi-objective optimization, Hard real-time systems, Approximation, Metaheuristic algorithms, SPM allocation.

ACM Reference Format:
Shashank Jadhav and Heiko Falk. 2022. Approximating WCET and Energy Consumption for Fast Multi-Objective Memory Allocation. In *Proceedings of the 30th International Conference on Real-Time Networks and Systems (RTNS '22), June 7–8, 2022, Paris, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534879.3534889

RTNS '22, June 7–8, 2022, Paris, France

ACM ISBN 978-1-4503-9650-9/22/06…$15.00
https://doi.org/10.1145/3534879.3534889

1 INTRODUCTION

Embedded systems with hard real-time constraints need to adhere to strict timing constraints. Failure to meet such constraints might lead to disastrous consequences. Therefore, it is necessary to consider WCET as an objective while designing such systems. Moreover, such embedded systems could be small, mobile, and battery-operated with a limited energy budget. Therefore, the energy expenditure is another necessary design objective to be considered.

Suhendra et al. [25] and Falk et al. [5] minimized the system's WCET using compiler-level SPM allocation. Steinke et al. [24] showed that placing program and data objects into SPM can improve energy consumption. WCET- and energy-critical parts of a program can be different. Therefore, we can design an SPM allocation-based multi-objective optimization problem, where WCET and energy consumption are simultaneous minimization objectives. For contradicting non-dominating WCET and energy solutions, a system designer can use the Pareto-optimal solution set to perform design space exploration. We can use population-based metaheuristic algorithms to iteratively search the solution space, solve the multi-objective optimization problem, and find optimal solutions trading multiple objectives.

To determine WCET, we can use a static analysis-based approach that is suitable to perform compiler-based WCET-aware optimization. Similarly, the energy consumption of a program can be estimated using an energy model and combining it with the timing behavior of the program. In this paper, we will use the tight integration of `aiT` [1] and `EnergyAnalyser` [26] with the compiler to statically calculate WCET and energy consumption of the program at compile-time, respectively. Depending on the program size, performing WCET and energy analyses using these analyzers can be time-consuming. While performing metaheuristics-based multi-objective optimization, analyzing all the individuals in the population using such analyzers can become time-exhaustive. Therefore in this paper, we propose a novel approximation model to approximate objective values and use it instead of the time-exhaustive analyzers to analyze the population. The approximation model proposed in this paper is built based on the data collected from four initial analyzer-based WCET and energy analyses of the program with different code allocations. The approximation model approximates WCET and energy objectives to bring the metaheuristic algorithm to fruition. Finally, the compiler performs safe analyses on the final solution set using the static analyzers to present correct WCET and energy values.

In this paper, we perform multi-objective optimization using two metaheuristic algorithms, i.e., Flower Pollination Algorithm (FPA) [27] and Strength Pareto Evolutionary Algorithm (SPEA) [28].

During evaluations, we perform multi-objective optimization using the approximation model proposed in this paper. For the sake of brevity, we refer to this optimization run as MO_{approx} in this paper. Furthermore, using the WCET and energy analyzers, we also perform multi-objective optimization referred to as MO_{eval} in this paper. We compare the results from MO_{approx} and MO_{eval} in terms of compilation time, approximation error, and the quality of obtained final solution set. The key contributions of this paper are:

- We proposed an approximation model to approximate the WCET and the energy consumption of a program.
- We performed SPM-allocation-based multi-objective optimization in conjunction with the proposed approximation model.
- Using MO_{approx}, we achieved on average 94.12% decrease in compilation time compared to MO_{eval}.
- The approximation errors during MO_{approx} run were in the acceptable range of 2% - 4% on average.
- The quality of the solutions provided by MO_{approx} are comparative to the ones from MO_{eval}.

This paper is organized as follows: Section 2 provides an overview of the related work. Section 3 briefly describes the compiler framework within which the approximation model and the multi-objective optimization is implemented. Section 4 discusses the SPM allocation-based multi-objective optimization problem solved in this paper. Section 5 presents models for approximation of the objective functions. Section 6 briefly discusses the metaheuristic algorithms to solve MO_{eval} and MO_{approx}. Section 7 presents the evaluation results.

2 RELATED WORK

SPM allocation-based approaches proposed in the past are in the context of single-objective Average-Case Execution Time (ACET)- or WCET- or energy-aware optimization [5, 13, 23]. Suhendra et al. [25] proposed an approach to perform WCET-aware static data SPM allocation using ILPs. Falk et al. [7] modified and proposed an approach to perform WCET-aware static program code SPM allocation using ILPs. Steinke et al. [24] proposed an approach to perform static energy-aware SPM allocation. To reduce the total energy consumption of the system, Ishitobi et al. [11] proposed an algorithm to find optimal code placement for SPM as well as for cacheable and non-cacheable regions. To find the optimal code layout that minimizes energy consumption, Balasundaram et al. [2] proposed a metaheuristics-based optimizer. These approaches used either ILP-, heuristic-, or evolutionary-based techniques focusing on optimizing a single code property without considering its impact on the other.

Multi-objective optimizations are rarely exploited to perform compiler-level optimization. Hoste et al. [9] proposed an evolutionary algorithm-based multi-objective optimization framework to explore various compiler optimization levels that automatically find Pareto-optimal optimization levels. A stochastic evolutionary approach to find Pareto optimal compiler optimization sequences for a bi-objective problem was proposed by Lokuciejewski et al. [15]. Muts et al. [17] proposed a function-inlining-based multi-objective optimization. Jadhav et al. [12] proposed a compiler-level multi-objective optimization framework using FPA.

ILP-based approaches present a set of limitations when handling multiple objectives simultaneously. They work best for single-objective or multi-criteria optimization problems. ILP solvers like, e.g., Gurobi, provide features to break down multi-objective problems and formulate them as primary and secondary objectives problems. Besides, ILPs are NP-hard [14] and can require exponentially high compilation times. Furthermore, all these evolutionary single- and multi-objective optimization approaches perform evaluations using some respective analyzer, which could be a compilation time bottleneck and time-exhaustive in real-world scenarios. Consequently, in this paper, we proposed an approximation model to approximate WCET and energy objectives while performing multi-objective optimization.

Machine learning-based approaches have been exploited to predict the WCET and energy consumption of the code. Huybrechts et al. [10] proposed a machine learning-based hybrid approach to predict WCET using different regression models. Bonenfant et al. [3] predicted WCET at the source code level using machine learning techniques in combination with static source code analysis. Sachan et al. [22] determined optimal compiler optimization setting to minimize energy consumption using power profile results, real hardware performance characteristics, and machine learning models. The accuracy of these approaches depends on the number of evaluations performed to collect the data and build the machine learning model. Also, it could depend on the proper tuning of the hyperparameters used for the respective machine learning models. In our approach, in contrast, we use a fixed number of evaluations, i.e., four, to build the approximation model and do not depend on any hyperparameters.

3 WCET-AWARE C COMPILER FRAMEWORK (WCC)

The WCET-Aware C Compiler (WCC) [6], as the name suggests, is a C compiler that can perform sophisticated WCET- and Energy-aware code analyses and optimizations. Figure (1) represents the structure of the WCC framework. WCC supports Cortex-M0 and ARM7TDMI from ARM and TriCore TC1796 and TC1797 processors from Infineon. WCC comprises a parser, a high-level C-like intermediate representation called ICD-C, a code selector, a low-level assembler-like intermediate representation called ICD-LLIR, and a code generator.

WCC parses a user-annotated ANSI-C source code into the ICD-C representation and further transforms it using a code selector into the low-level ICD-LLIR representation. The user-provided annotations, called flow facts, provide information such as loop iteration counts, recursion depths, etc. This information is necessary during WCET and energy analyses.

The integration of a static WCET analyzer tool called aiT [1] and an energy analysis tool called EnergyAnalyser [26] within WCC enables WCC to perform WCET and energy analyses and optimization at compile time.

At both ICD-C and ICD-LLIR representations, we can perform various compile-time optimizations. For example, optimizations like loop unrolling and function inlining can be carried out at the ICD-C level, whereas we can carry out optimizations like register and memory allocation at the ICD-LLIR level. Furthermore, WCC

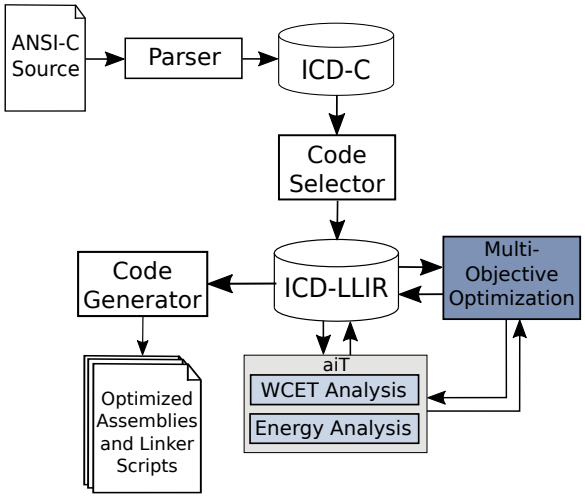

Figure 1: WCC Framework

is also equipped with other optimizations to deal with several objectives like WCET, energy, code size, schedulability, etc. It can perform ILP-based optimizations to deal with these objectives. An optimal solution is deterministically provided that fulfills the given constraints by using an ILP model. Moreover, WCC is equipped to deal with multi-tasking systems [16] and multi-core systems [19].

Static memory allocation optimization, specifically static SPM allocation considered in this paper, is a low-level optimization technique. It requires both high-level flow fact-based information and low-level target architecture knowledge. To perform multi-objective optimization in this paper, we use metaheuristic algorithms, namely FPA and SPEA. The framework integrated within WCC to support metaheuristic algorithms is explained, in detail, in further sections.

4 OPTIMIZATION PROBLEM

Generally, real-world optimization problems deal with multiple optimization objectives. More often than not, these objectives contradict each other. Therefore, it is usually impossible to optimize all optimization objectives simultaneously. Such optimization problems are called multi-objective optimization problems. Contrary to single-objective optimization problems, these problems can have multiple solutions resulting in a trade-off between optimization objectives.

Finding optimal solutions using deterministic algorithms is challenging for a particular class of (NP-hard) optimization problems. Heuristic approaches, such as greedy algorithms or hill-climbing algorithms, can be exploited to find solutions to such problems. But, in the case of multi-objective problems, these heuristics might become sub-optimal. They tend to get trapped in regions of the search space to find local optimal solutions. In such scenarios, metaheuristics with problem-independent search strategies come into play.

Metaheuristic algorithms employ iterative approaches with mechanisms for escaping local optima and resume search in a different region of the search space. Evolutionary Algorithms (EA), a class of stochastic optimization methods that emulate the natural evolution process, are considered under the family of metaheuristics. Furthermore, nature-inspired algorithms, mostly swarm intelligence-based, are another class of metaheuristic algorithms. Due to their flexibility and simplicity, a wide range of applications employ these algorithms.

Let us first introduce the following definitions and notation used by us in regard to metaheuristic algorithms in this paper.

Definition 4.1. The set of all possible decision variables is called as the search space (X) of the multi-objective optimization problem.

Definition 4.2. The set of all possible objective values on the given search space is called as the objective space (Θ) of the multi-objective optimization problem, i.e.,

$$\Theta = \{F(x) | x \in X\}$$

Definition 4.3. For a minimization problem, a decision variable $x \in X$ dominates $y \in X$ ($x \prec y$) if, for all objectives, the objective values at x is less or at least equal to the objective value at y, i.e.,

$$\forall m \in \{1, 2, \ldots, q\} \quad F_m(x) \leq F_m(y)$$

and there exists at least one objective value at x that is better than at y, i.e.,

$$\exists n \in \{1, 2, \ldots, q\} \quad F_n(x) < F_n(y)$$

Definition 4.4. The decision variables or solutions $x \in X$ that are not dominated by any other solution in the objective space Θ are called Pareto optimal solutions.

Definition 4.5. The set of all Pareto optimal solutions is called as the Pareto optimal set, and the corresponding objective vectors form the Pareto optimal front.

In this paper, we formulate a compiler-level multi-objective optimization problem that performs SPM allocation to minimize the WCET and energy consumption of a program. SPM allocation is a compiler optimization where we can move basic blocks from slow Flash memory to SPM. The reduction in WCET and energy consumption can be achieved by optimally placing basic blocks in SPM. A multi-objective optimization problem performing SPM allocation (MO_{eval}) can be mathematically formulated as a minimization problem as follows.

$$\begin{aligned} \min_{x} \quad & F(x) = (F_1(x), F_2(x)) \\ \text{subject to} \quad & g(x) \leq 0 \end{aligned} \tag{1}$$

The function $F(x)$ maps a search space $X \subset \{0,1\}^d$ to an objective space $Z \subset \mathbb{R}^2$. $F_1(x)$ and $F_2(x)$ represent WCET and energy objectives, respectively. The search space X consists of $x \in X$, a d-dimensional binary decision variable vector, that represents the decision of whether a basic block b_j is placed in SPM or Flash, i.e., $x_j \in \{0, 1\}$ and $j = \overline{1, d}$, where d represents the total number of basic blocks. The constraint condition $g(x) \leq 0$ for the SPM allocation problem is the size of the SPM, i.e.,

$$g(x) = \sum_{j=1}^{d} B_j x_j - S_{SPM} - \sum_{j=1}^{d-1} s_j |x_j - x_{j+1}| \leq 0 \tag{2}$$

where B_j is the code size of basic block b_j, S_{SPM} is the total size of the SPM, and the third term represents the jump correction cost.

Performing SPM allocation can change the physical successors and predecessors of basic blocks, alter memory addresses, and invalidate previous jumps between them. To repair instructions, we need to add a jump correction code and ensure the proper control flow of the program. Depending on the type of jump between basic blocks, the requirement of the additional jump correction code may vary. $|x_j - x_{j+1}|$ determines whether extra jump correction cost is needed, and s_j is an architecture-specific term representing the cost in terms of jump code size. For the sake of brevity, we are omitting the modeling aspects of the jump correction code as it has been discussed in detail by Oehlert et al. [18]. For ARM Cortex-M0, s_j is modeled as follows.

$$s_j = \begin{cases} 2, \text{ if } b_j \text{ has } \textit{indirect\_call} \text{ or } \textit{explicit\_branch}, \\ 3, \text{ otherwise} \end{cases} \tag{3}$$

In this paper, we use metaheuristic algorithms that employ problem-independent approaches to solve MO_{eval}. Metaheuristic algorithms being iterative population-based algorithms, the evaluation of all individuals at every iteration is necessary to drive the optimization to fruition. These individuals in the population represent the decision variables from the search space. However, WCET and energy analyses are time-consuming processes. Therefore, given the large number of evaluations needed, finding approximated Pareto-optimal solutions to MO_{eval} even for small programs could take a long time.

To tackle this issue in this paper, we instead formulate an approximated multi-objective optimization problem (MO_{approx}) as follows.

$$\begin{aligned} \min_{x} \quad & \widetilde{F}(x) = (\widetilde{F_1}(x), \widetilde{F_2}(x)) \\ \text{subject to} \quad & g(x) \leq 0 \end{aligned} \tag{4}$$

where $\widetilde{F_1}(x)$ and $\widetilde{F_2}(x)$ represent approximated WCET and approximated energy minimization objectives, respectively. The constraint condition for MO_{approx} remains the same as presented in Equation (2). To solve MO_{approx}, instead of performing WCET and energy analyses using respective analyzers, we evaluate the individuals by approximating the objective values. We present the approximation model for said objectives in the following section.

5 APPROXIMATION MODEL

In this section, we propose an approximation model necessary to approximate the objectives while performing multi-objective optimization. To build the approximation model, we first generate four distinct individuals referring to four specific code allocation strategies explained in detail below. The analysis of these four individuals using WCET and energy analyzers provides us with all the data necessary to further approximate WCET and energy objectives. The approximation model proposed in this section is only used to iteratively analyze individuals from populations while running metaheuristic algorithms. Once a metaheuristic algorithm provides the final Pareto optimal solution set, we analyze the Pareto optimal individuals using WCET and energy analyzers to avoid minor approximation errors in the final output. This step ensures that WCC finally provides appropriately analyzed Pareto optimal solutions.

Let $\mathcal{M}, \mathcal{N}, \mathcal{O}, \mathcal{P} \in \mathcal{I}$ represent four individuals necessary to build the approximation model, and $\mathcal{I}$ is a set of all individuals. Individuals $\mathcal{M}$ and $\mathcal{N}$ represent the case where all basic blocks are placed in Flash and in SPM, respectively. As SPMs are small and fast memories, the size of SPM is hypothetically increased to place all basic blocks in SPM for individual $\mathcal{N}$. For individual $\mathcal{O}$, the first basic block is placed in Flash, the second basic block in SPM, and so on, i.e., each basic block is placed in the Flash memory and SPM in an alternating fashion starting with Flash memory. Contrary, for individual $\mathcal{P}$, the first basic block is placed in SPM, the second basic block in the Flash memory, and so on, i.e., each basic block is placed in Flash memory and SPM in an alternating fashion starting with SPM.

These four individuals are analyzed using WCET and energy analyzers, and the objective values of respective individuals are annotated back within WCC at basic block level granularity. Within the approximation model, the individual $\mathcal{M}$ is used to approximate the objective values of basic blocks placed in Flash memory, whereas the individual $\mathcal{N}$ to approximate the objective values of basic blocks placed in the SPM. The alternating placement of basic blocks within individuals $\mathcal{O}$ and $\mathcal{P}$ approximates the effect of jump correction on the basic block's objective value, i.e., the analysis of individuals $\mathcal{O}$ and $\mathcal{P}$ provides the extra cost in terms of WCET and energy consumption required to jump between Flash memory and SPM. If a basic block has more than one successor and the particular case of basic block allocation is not represented by these four individuals, the approximation model uses architecture-specific details to approximate the extra cost needed in terms of WCET and energy consumption. In this paper, we have considered the ARM Cortex-M0 architecture during evaluations. The proposed approximation model can be extended for other processors just by integrating the architecture-specific details needed for the approximation of the jump correction costs. The mathematical formulation of the approximation model is explained in detail next.

Let $W_{i,j}$ and $E_{i,j}$ represent WCET and energy values of basic block b_j of individual $i \in \mathcal{I}$. Let E represent the set of edges of the program's *control flow graph*, where $e_{j,k} \equiv (b_j, b_k) \in E$ indicates that basic block b_k is reached from basic block b_j.

Equation (5) represents the proposed model for approximating WCET. The first and the second term calculates the WCET of all basic blocks placed in Flash and in SPM, respectively. $W_{\mathcal{M},j}$ and $W_{\mathcal{N},j}$ represent the WCETs of the j^{th} basic block of individuals $\mathcal{M}$ and $\mathcal{N}$, respectively. As mentioned before, $x_j \in \{0, 1\}$ represents a decision variable for basic block b_j, and $\overline{x_j} = \neg x_j$.

$$\begin{aligned} \widetilde{F_1}(x) = & \sum_{j=1}^{d} W_{\mathcal{M},j}\overline{x_j} + \sum_{j=1}^{d} W_{\mathcal{N},j}x_j + \\ & \sum_{j=1}^{d-1} (W_{O,j} - (W_{\mathcal{M},j}\overline{x_j} + W_{\mathcal{N},j}x_j))\tau_{O_j} + \\ & \sum_{j=1}^{d-1} (W_{\mathcal{P},j} - (W_{\mathcal{M},j}\overline{x_j} + W_{\mathcal{N},j}x_j))\tau_{\mathcal{P}_j} + \\ & \sum_{j=1}^{d-1} \mathcal{H}_j\eta_j + \sum_{j=1}^{d-1} \mathcal{L}_j\delta_j \end{aligned} \tag{5}$$

In Equation (5), the third and fourth terms calculate additional jump correction costs incurred when succeeding basic blocks are in different memory. $W_{O,j}$ and $W_{\mathcal{P},j}$ represents the WCETs of the j^{th} basic block of individuals O and $\mathcal{P}$, respectively. If the j^{th} basic block of individual O is in SPM, then the same basic block of individual $\mathcal{P}$ will always be in the Flash. To guarantee that we add the correct jump correction cost to our approximation model, we multiply the third and fourth terms by τ_{O_j} and $\tau_{\mathcal{P}_j}$, respectively.

$$\tau_{O_j} = \begin{cases} 1, \text{ if } x_j = o_j \ \& \\ \qquad x_{j+1} = o_{j+1}, \\ 0, \text{ otherwise} \end{cases} \tag{6}$$

$$\tau_{\mathcal{P}_j} = \begin{cases} 1, \text{ if } x_j = p_j \ \& \\ \qquad x_{j+1} = p_{j+1}, \\ 0, \text{ otherwise} \end{cases} \tag{7}$$

τ_O and $\tau_{\mathcal{P}}$ are $(d-1)$-dimensional binary vectors, where $1's$ indicate that basic blocks b_j and b_{j+1} in x have the same configuration as in individuals O and $\mathcal{P}$, respectively.

In Equation (5), the fifth and sixth terms cover the cases where the basic block branches to basic blocks other than its immediate successor or calls another function. η is a $(d-1)$-dimensional binary vector, where $\eta_j = 1$ if there exists an edge $e_{j,k} \in E$ between the basic blocks b_j and b_k, the successor basic blocks b_{j+1} and b_k are different, and basic blocks b_j and b_k are in different memories, i.e.,

$$\eta_j = \begin{cases} 1, \text{ if } b_k \neq b_{j+1} \ \& \ (b_j, b_k) \in E \ \& \ x_j \neq x_k, \\ 0, \text{otherwise} \end{cases} \tag{8}$$

In case of δ, a $(d-1)$-dimensional binary vector, the basic blocks b_j and b_k are in the same memory, i.e.,

$$\delta_j = \begin{cases} 1, \text{ if } b_k \neq b_{j+1} \ \& \ (b_j, b_k) \in E \ \& \ x_j = x_k, \\ 0, \text{otherwise} \end{cases} \tag{9}$$

$\mathcal{H}_j$ and $\mathcal{L}_j$ represent extra jump correction cost incurred due to the jump between basic blocks b_j and b_k across different and same memories, respectively, i.e.,

$$\mathcal{H}_j = C_j h_j \quad \text{and} \quad \mathcal{L}_j = C_j l_j \tag{10}$$

where C_j represents the worst-case execution count of basic block b_j obtained from the data collected during the initial WCET analysis. h_j and l_j are architecture-specific terms representing the extra cycles required to perform a single jump across different memories and within the same memory, respectively. For ARM Cortex-M0, $l_j = 1$ cycle and h_j is modeled as follows

$$h_j = \begin{cases} 3, \text{ if } b_j \text{ has } \textit{indirect\_call} \text{ or } \textit{explicit\_branch}, \\ 5, \text{ otherwise} \end{cases} \tag{11}$$

The final decision variable in the solution refers to the final basic block exiting the program. Therefore, we do not consider the final basic block in the summation except for the first and second term in Equation (5) as it will not incur any additional jump correction cost.

$$\begin{aligned} \widetilde{F_2}(x) = & \sum_{j=1}^{d} E_{\mathcal{M},j}\overline{x_j} + \sum_{j=1}^{d} E_{\mathcal{N},j}x_j + \\ & \sum_{j=1}^{d-1}(E_{O,j} - (E_{\mathcal{M},j}\overline{x_j} + E_{\mathcal{N},j}x_j))\tau_{O_j} + \\ & \sum_{j=1}^{d-1}(E_{\mathcal{P},j} - (E_{\mathcal{M},j}\overline{x_j} + E_{\mathcal{N},j}x_j))\tau_{\mathcal{P}_j} + \\ & \sum_{j=1}^{d-1}\mathcal{E}_{\mathcal{H}j}\eta_j + \sum_{j=1}^{d-1}\mathcal{E}_{\mathcal{L}j}\delta_j \end{aligned} \tag{12}$$

A similar approximation strategy is employed to approximate the energy objective. Equation (12) represents the proposed model for approximating energy consumption. The purpose of each term in Equation (12) is similar to the terms in Equation (5). Here instead of using the WCETs ($W_{i,j}$), we use the energy values ($E_{i,j}$) of $\mathcal{M}, \mathcal{N}, O$, and $\mathcal{P}$. The terms in equations (6) - (9) are used in Equation (12) to approximate the energy objective. The only change occurs in the fifth and the sixth term, where $\mathcal{E}_{\mathcal{H}j}$ and $\mathcal{E}_{\mathcal{L}j}$ represent the extra energy cost incurred due to the jump between basic blocks b_j and b_k, i.e.,

$$\mathcal{E}_{\mathcal{H}j} = \mathbb{C}_j s_j e_{h_j} \quad \text{and} \quad \mathcal{E}_{\mathcal{L}j} = \mathbb{C}_j s_j e_{l_j} \tag{13}$$

where $\mathbb{C}_j$ represents the execution count of basic block b_j obtained from initial energy analysis, and s_j is modeled in Equation (3). e_{h_j} and e_{l_j}, architecture-specific terms derived from the energy model of the respective architecture, represent the cost in terms of energy required to perform a single jump across different memories and within the same memory, respectively. For ARM Cortex-M0, $e_{h_j} = 2989$ and $e_{l_j} = 1441$.

6 MULTI-OBJECTIVE OPTIMIZATION

In this section, we exploit the metaheuristic algorithms to perform and solve MO_{approx}. Figure (2) shows the multi-objective optimization process and the structure of the metaheuristic algorithm library within WCC. In the case of MO_{eval}, the evaluation step uses WCET and energy analyzers integrated within WCC. We represent this step within the figure with a dotted line. Whereas MO_{approx} uses the proposed approximation model during evaluation, which we represent using a solid line within the figure.

Algorithm (1) describes the pseudo-code for performing MO_{approx}. We first initiate the optimization problem using the search space (X), the objective space (Z), optimization constraint (Equation (2)), and minimization objectives ($\widetilde{F}(x)$). We then generate four individuals $\mathcal{M}, \mathcal{N}, O$, and $\mathcal{P}$ as described in Section 5. We evaluate these individuals using the WCET and energy analyzers integrated within the compiler, and then we collect the related data for these individuals. Eventually, we use this data to approximate the WCET and energy objective using equations (5) and (12).

To solve MO_{approx}, we first initialize the initial population randomly, check if it satisfies the SPM size constraint, and evaluate them using the approximation model. For individuals not satisfying Equation (2), we repair the individual by randomly removing one basic block at a time from SPM until the constraint is satisfied.

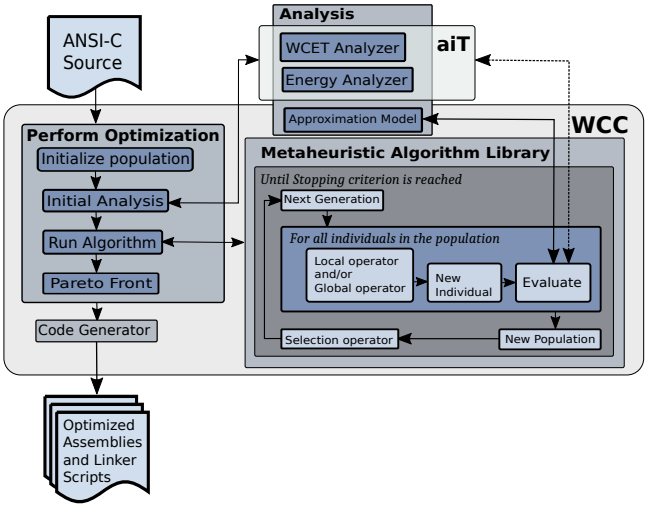

Figure 2: Multi-Objective Optimization Framework within WCC

We use this repair mechanism to avoid the evaluations of invalid individuals. At every iteration, we update each individual with the help of global or local operators. The operators vary depending on the metaheuristic algorithm under consideration. At every iteration, a selection operator chooses evaluated individuals with Pareto-optimal objective values for the next generation. The appropriate stopping criteria are usually predefined, for example, the maximum number of generations. After the algorithm meets the stopping criteria, the compiler outputs a set of approximated Pareto optimal solutions from the final generation of the population.

In this paper, we use two metaheuristic algorithms, SPEA and FPA, to solve the compiler-level multi-objective optimization. SPEA is a well-established and efficient algorithm that falls into the sub-category of Evolutionary Algorithms (EA) within the large collection of metaheuristic algorithms. Zitzler [29] showed that SPEA outperforms other EA such as NSGA, VEGA, etc. Therefore, we apply the SPEA algorithm from the EA sub-category of the metaheuristic algorithm to solve the proposed multi-objective optimization. FPA is a relatively recently proposed, simple, and flexible algorithm that falls into the sub-category of Nature-Inspired Algorithms (NIA) within metaheuristic algorithms. Yang [27] showed that FPA obtains better results compared to other multi-objective algorithms such as VEGA, NSGA-II, MODE, and also SPEA. Therefore, we applied both FPA and SPEA, algorithms that fall in different sub-categories of metaheuristic algorithms, to solve our multi-objective optimization.

We consider the choice of metaheuristic algorithms, a performance comparison between these two algorithms, and fine-tuning of the algorithmic parameters as out of the scope of this work, therefore, we only briefly describe SPEA and FPA algorithms. SPEA is a population-based algorithm that uses selection, recombination, and mutation techniques iteratively to find approximated Pareto-optimal solutions. SPEA uses an external set to store non-dominated individuals from the population. It assigns scalar fitness values to individuals from the population at each generation using Pareto

Algorithm 1 Approximated Multi-Objective SPM Allocation

Input: Evaluated individuals $\mathcal{M}, \mathcal{N}, \mathcal{O}$, and $\mathcal{P}$,
    approximate objective functions $\widetilde{F}_1(x)$ and $\widetilde{F}_2(x)$.
Output: Approximated Pareto-optimal solutions.
Randomly initialize initial population.
Repair individuals from the population if necessary.
Evaluate initial population using Equations (5) and (12).
while Stopping criteria is not reached **do**
    Update Individual using **Local** operator.
    Update Individual using **Global** operator.
    Evaluate individual using Equations (5) and (12).
    Update to next generation using **Selection** operator.
end while
return Approximated Pareto-optimal solutions.

dominance. SPEA performs recombination (Local operator from Algorithm (1)) using single-bit recombination and mutation (Global operator from Algorithm (1)) using multi-bit flip mutation. It uses a clustering technique to reduce the number of individuals in the external set to maintain the selection pressure.

FPA mimics the behavior of the pollination process seen in the flowering plants. It uses two different operators to search for solutions in the local and the global solution space. The local pollination operator (Local operator from Algorithm (1)) mimics the local pollination process to perform a local random walk and searches for better solutions in the local neighborhood. Furthermore, FPA uses a global pollination operator (Global operator from Algorithm (1)) to mimic the behavior of pollinators to perform the global pollination process. To mimic this behavior, FPA uses a Lévy flight distribution [20], which imitates the characteristics of the pollinator's flight pattern over long distances. It helps FPA to exit local neighborhoods and to continue to search elsewhere within the search space. FPA uses a switch-probability to switch between local and global pollination operators.

7 EVALUATIONS

In this section, we perform evaluations to compare MO_{approx} and MO_{eval} in terms of the compilation times required to find the final approximated Pareto front, the approximation error, and the quality of the obtained solutions. The evaluations are conducted using the WCC compiler framework for the ARM Cortex-M0 architecture. All optimization problems were solved using a Ubuntu 18.04 LTS server featuring four Intel XEON Gold 6146 processors, each with 12 cores and 24 threads clocked at 3.2 GHz and 1.48 TB of RAM. The static WCET and energy analyses for MO_{eval} are performed using `aiT` v21.04i and `EnergyAnalyser` v21.04i, respectively. The initial four individuals in MO_{approx} are analyzed using the WCET and energy analyzers to build the approximation model, the rest of the optimization run uses the proposed approximation model, and the obtained final Pareto solutions are again safely analyzed using the WCET and energy analyzers.

We use the MRTC benchmark suite [8] with loop bound annotations from TACLeBench project [4] for evaluation purposes. While performing evaluations, we use *-O0* optimization flag to turn off any other compiler-level optimization. We adjust the SPM size

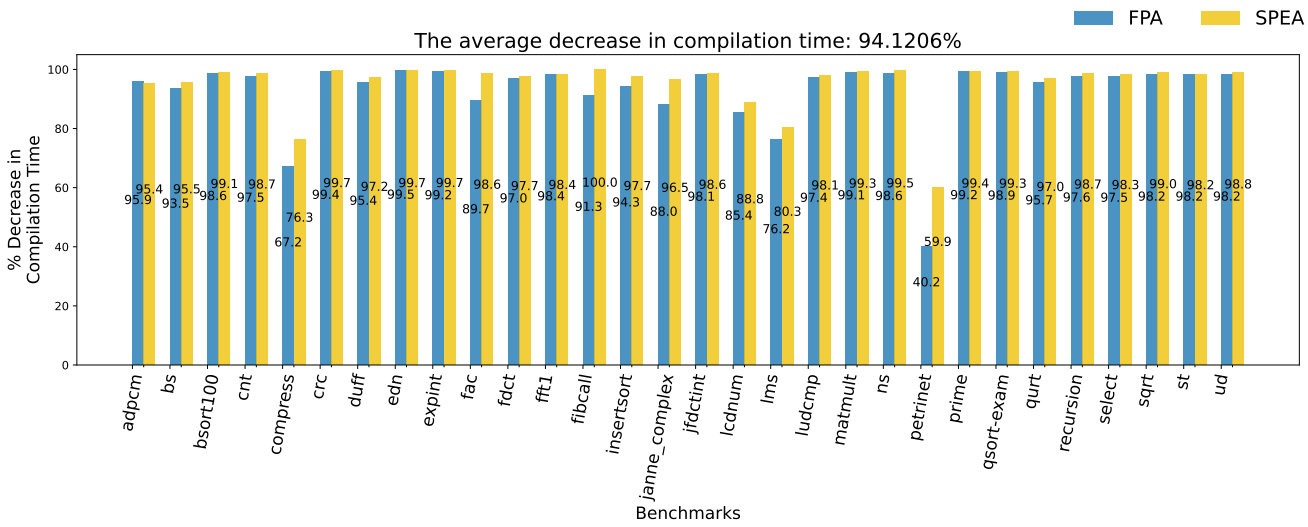

Figure 3: % decrease in the compilation times achieved by using approximated multi-objective optimization.

individually to 40% relative to the benchmark size to increase the pressure on the optimization even for smaller benchmarks.

The initial population of any metaheuristic algorithm is defined randomly and can influence the final obtained solutions. Therefore, we repeat the evaluations for both FPA and SPEA using five different initial populations to reduce the influence of the initial population on the final results. In the end, we combine all the individual Pareto fronts to find a final Pareto front. We set a timeout value of 20 h for each benchmark, and the compiler terminates the compilation if the optimizations have not finished successfully.

While solving MO_{eval} and MO_{approx} by using FPA, we used the following parameter values: as the probability of local pollination is always higher compared to global pollination in nature, the switch probability between the local and the global pollination operators is set to $p_s = 0.8$. According to parametric studies [21, 27], the positive integer for the standard gamma function $\lambda = 1.5$, and the scaling factor $\gamma = 0.1$ work well for most of the cases. We considered the size of the population as 10, the first stopping criterion - the maximum number of generations - is set to 80, and the second stopping criterion - the maximum number of generations for which the population remains the same - is set to 10.

For SPEA-based multi-objective optimization, we used the following parameter values: The size of the population for each generation and the size of the external population set is fixed to 10. As the probability of crossover is higher when compared to mutation during evolution, we set the crossover probability for SPEA as 0.8 and the mutation probability as 0.2. We use a single-point crossover while running SPEA. Using two d-dimensional parent individuals, a single-point crossover takes a union of the front m entries of the first parent individual and the rear $d - m$ entries of the second parent individual to create a new child individual, where $m \in [0, d-1]$ is chosen randomly. Furthermore, we used a multi-bit mutation strategy that mutates multiple bits in an individual randomly to create a new mutated individual. The results obtained during these evaluations are valid for the algorithm parameters described above.

7.1 Compilation Times

Figure (3) shows a percentage decrease in the compilation times achieved by using MO_{approx} when compared with MO_{eval}. The x-axis of the figure shows the evaluated benchmarks, and the y-axis of the figure represents the percentage decrease in the compilation times. The legend at the top-right corner of the figure represents the metaheuristics used during evaluation, i.e., FPA and SPEA. The values for percentage decrease in the compilation times for each benchmark are displayed on the respective bars.

As `petrinet` within the MRTC benchmark suite is relatively tiny, performing WCET and energy analyses is relatively quick. Therefore, the decrease in compilation times achieved is 40.2% and 59.9% for FPA and SPA, respectively, which is relatively less than other benchmarks. Using FPA algorithm, on average MO_{approx} achieved 92.79% decrease in compilation times compared to MO_{eval}. Using SPEA algorithm, on average MO_{approx} achieved 95.45% decrease in compilation times compared to MO_{eval}.

Finally, we achieved a significant percentage decrease in compilation times of 94.12% on average by using the proposed approximation model over the traditional WCET and energy analyzers while performing multi-objective optimization.

Furthermore, we calculated the sum of all the compilation times for all the benchmarks, and the total time taken by MO_{eval}–FPA and MO_{eval}–SPEA optimization runs are 63.83 h and 78.37 h, respectively, while MO_{approx}–FPA and MO_{approx}–SPEA took just 3.68 h and 5.34 h, respectively.

7.2 Pareto Fronts

Figure (4) shows the solutions found by FPA and SPEA using both MO_{approx} and MO_{eval}. For the sake of brevity, this subsection

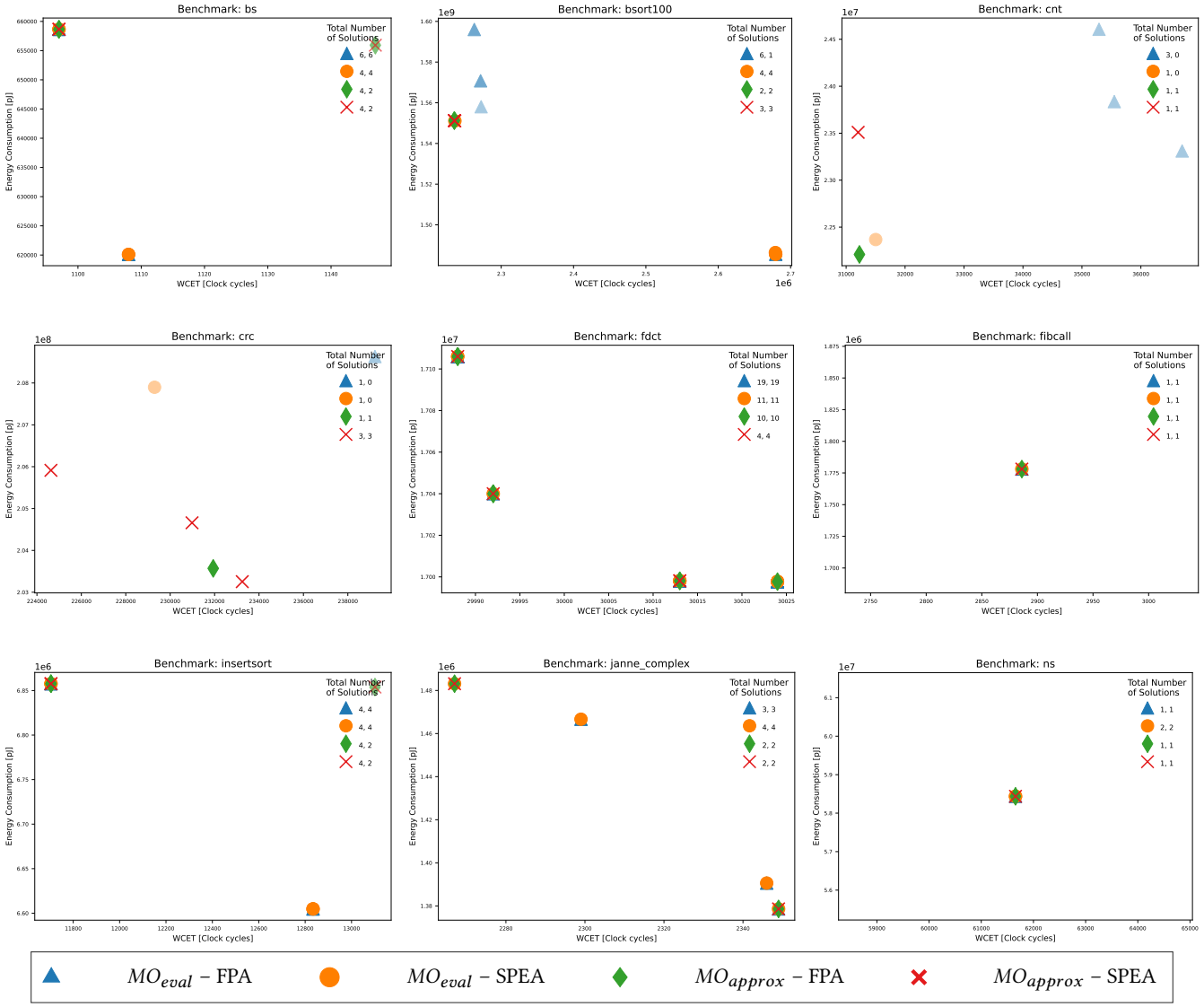

Figure 4: Solutions Obtained by Metaheuristics-based Multi-Objective Optimization

presents the solutions for only nine benchmarks[1]. In this figure, we represent the final Pareto fronts using a 2D scatter plot. The x-axis and the y-axis represent WCET and energy objectives, respectively. The title of each plot is the benchmark name. The legend at the bottom of the figure represents the approach to which a particular solution belongs. The legend at the top-right corner of each plot shows the total number of solutions returned by the respective approach. Moreover, the same legend shows the number of solutions out of the total solutions on the final Pareto front. The final Pareto front ($\mathcal{P}$) represents the set of indifferent solutions from the union of the Pareto optimal solutions obtained from all the approaches. In the figure, the darker colors represent the solutions on $\mathcal{P}$ and the fainter version of those colors represents the solutions returned by each approach. As we considered a minimization problem, the solutions on the bottom-left corner of the plots generally represent the final Pareto front.

For `bs` benchmark, we can see that all four approaches were able to find solutions on $\mathcal{P}$. But, all the solutions obtained by MO_{eval} lie on the final Pareto front $\mathcal{P}$ and only two solutions out of four obtained by MO_{approx}–FPA and MO_{approx}–SPEA lie on $\mathcal{P}$. A similar observation can be made in the case of `insertsort` benchmark, where all solutions obtained by MO_{eval} lie on $\mathcal{P}$ and two out of four solutions obtained by MO_{approx} lie on the final Pareto front $\mathcal{P}$. However, MO_{approx} was able to find these solutions for `bs` and `insertsort` in 94.9% and 96.39% less compilation times on average.

For `bsort100` benchmark, we can see that all four approaches were able to find solutions on $\mathcal{P}$. But, MO_{eval}–FPA contributed

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

to only one solution, while from others, all the solutions are on $\mathcal{P}$. MO_{approx}–SPEA and MO_{eval}–SPEA were able to find 4 and 3 solutions, respectively, on $\mathcal{P}$ that are distinct in the solution space but mapped to two points in the objective space. We can see a similar case for `fdct` benchmark, where multiple distinct solutions in the solutions space map to four final Pareto points in the objective space.

In case of `cnt` and `crc`, MO_{approx} was able to find better solutions using both FPA and SPEA. For `fibcall` benchmark, we can see that there exists only one solution on $\mathcal{P}$, and in all four cases, we obtained the same solution. For `ns` benchmark, MO_{eval}–SPEA obtained two distinct solutions in the search space that map to the same point in the objective space. All other approaches found a single solution in the search space that maps to that same point in the objective space.

In the case of `fdct` and `janne_complex`, the total number of solutions obtained by MO_{approx} is less compared to MO_{eval}, but all of them are on $\mathcal{P}$. However, MO_{approx} was able to find these solutions for `fdct` and `janne_complex` in 97.35% and 92.25% less compilation times on average. The approximation error incurred due to the use of the proposed approximation model is evaluated in the next subsection.

7.3 Approximation Error

In this subsection, we perform MO_{approx} optimization run to calculate the approximation error by using the percentage mean l_1 error. We evaluated each individual simultaneously using the approximation model and the respective analyzers to calculate the said error. Let $v = \{v^1, \ldots, v^N\}$ be the objective values evaluated by using the analyzer, $\widetilde{v} = \{\widetilde{v}^1, \ldots, \widetilde{v}^N\}$ be the objective values approximated by approximation model, and N is the total number of individuals over all the populations and generations of the algorithm. Then, the percentage mean l_1 error is defined as

$$ME_p^{l_1} = \frac{1}{N}\sum_{q=1}^{N}\frac{|\widetilde{v}^q - v^q|}{v^q} * 100 \tag{14}$$

Using this equation, we calculate approximation errors for all benchmarks.

The `expint` benchmark returned a maximum approximation error of 10.04% for WCET, and `recursion` returned a maximum approximation error of 8.85% for energy. For `expint` benchmark, MO_{eval}–SPEA and MO_{approx}–SPEA found one solution each on the final Pareto front, and for `recursion` benchmark, three solutions lie on the final Pareto front all found by MO_{eval}–SPEA. Furthermore, `fdct` returned a minimum approximation error of 0.13% for WCET, and the same returned a minimum approximation error of 0.12% for energy.

The proposed approximation model for WCET returned on average 3.92% and 3.95% mean l_1 error over all benchmarks for FPA and SPEA, respectively. The proposed approximation model for energy returned on average 2.2% and 3.12% mean l_1 error for FPA and SPEA, respectively.

As mentioned before, the approximation model is only used to iteratively analyze the individuals within the populations generated during the run of the metaheuristic algorithm. The final Pareto optimal population set outputted by the metaheuristic algorithm is then evaluated using the WCET and energy analyzers integrated within WCC. The evaluation of the final Pareto optimal population using WCC's analyzers provide the user with properly analyzed binaries without minor approximation errors. Therefore, the approximation errors calculated during this evaluation run refer only to the evaluation of individuals generated only during the metaheuristic algorithm run using the proposed approximation model. The quality and the quantity of the solutions obtained during these evaluations are evaluated and interpreted in detail using quality metrics in the next subsection.

7.4 Quality Metrics

To evaluate and compare the quality of the proposed MO_{approx}, we use d ifferent quality metrics, namely *Coverage* (*C*), *Non-Dominated Ratio* (*NDR*), and *Non-Dominated Solutions* (*NDS*). Obtaining exact Pareto-optimal solutions to such a multi-objective optimization problem is ambitious. Therefore, under a realistic assumption that the exact Pareto front is unknown, we consider the final Pareto front ($\mathcal{P}$) discussed in subsection 7.2 as the best available approximation of the exact Pareto front to calculate the quality metrics. Further, we introduce necessary quality metrics definitions.

Definition 7.1. Coverage ($C \in [0, 1]$) describes the total number of dominated points in a solution set A, i.e.,

$$C = 1 - \frac{|\{a \in A : \exists p \in \mathcal{P}, a \preceq p\}|}{|A|} \tag{15}$$

If all the elements of $\mathcal{P}$ dominate A, then $C = 1$, and if all the elements of A dominate or are equal to the elements of $\mathcal{P}$, then $C = 0$. The lower the value of C, the better.

Definition 7.2. Non-Dominated Ratio ($NDR \in [0, 1]$) measures the ratio of non-dominated solutions that are contributed by a particular solution set A to the non-dominated solutions provided by all solutions sets, i.e.,

$$NDR = \frac{|\mathcal{P} \cap A|}{|\mathcal{P}|} \tag{16}$$

The higher the value of *NDR*, the better.

Definition 7.3. Non-Dominated Solutions ($NDS \in [0, 1]$) calculates the number of non-dominated solutions concerning the solution set A itself, when compared to $\mathcal{P}$, i.e.,

$$NDS = \frac{|a \in A : a \in \mathcal{P}|}{|A|} \tag{17}$$

The higher the value of *NDS*, the better.

Table (1) shows *C*, *NDR*, and *NDS* values for all the evaluated benchmarks. For each benchmark, the table presents the values of the quality metrics for both MO_{eval} and MO_{approx}. Both evaluation runs are performed using FPA and SPEA, therefore, the respective quality metrics for each algorithm are presented in the table. Under each quality metric column, we have compared their values, and for each benchmark, the better quality metric is highlighted in the table.

From overall evaluations, we see that MO_{eval} using FPA and SPEA found solutions with either better or indifferent *C* for 7 and 13 benchmarks, while MO_{approx} using FPA and SPEA found for 8 and 11, respectively. Therefore in terms of *C*, both approaches

Table 1: Performance Metrics for FPA and SPEA

	Coverage (*C*)				Non-Dominated Ratio (*NDR*)				Non-Dominated Solutions (*NDS*)			
	MO_{eval}		MO_{approx}		MO_{eval}		MO_{approx}		MO_{eval}		MO_{approx}	
Benchmarks	FPA	SPEA	FPA	SPEA	FPA	SPEA	FPA	SPEA	FPA	SPEA	FPA	SPEA
adpcm	1	**0.789**	0.8	1	0	**0.053**	**0.053**	0	0	0.053	**0.1**	0
bs	0.684	0.789	**0.6**	**0.6**	**0.316**	0.211	0.105	0.105	**0.316**	0.211	0.2	0.2
bsort100	**0.684**	0.789	0.8	0.7	0.053	**0.211**	0.105	0.158	0.053	0.211	0.2	**0.3**
cnt	1	1	**0.9**	**0.9**	0	0	**0.053**	**0.053**	0	0	**0.1**	**0.1**
compress	1	**0.474**	0.9	1	0	**0.263**	0.053	0	0	**0.263**	0.1	0
crc	1	1	0.9	**0.7**	0	0	0.053	**0.158**	0	0	0.1	**0.3**
duff	1	**0.684**	1	0.8	0	**0.211**	0	0.105	0	**0.211**	0	0.2
edn	1	1	1	**0.9**	0	0	0	**0.053**	0	0	0	**0.1**
expint	1	**0.895**	1	0.9	0	**0.053**	0	**0.053**	0	0.053	0	**0.1**
fac	**0.895**	**0.895**	1	1	**0.105**	**0.105**	0	0	**0.105**	**0.105**	0	0
fdct	**0**	0.421	**0**	0.6	**1**	0.579	0.526	0.211	**1**	0.579	**1**	0.4
fft1	1	1	1	**0.9**	0	0	0	**0.053**	0	0	0	**0.1**
fibcall	0.947	0.947	**0.9**	**0.9**	**0.053**	**0.053**	**0.053**	**0.053**	0.053	0.053	**0.1**	**0.1**
insertsort	0.789	0.789	**0.6**	**0.6**	**0.211**	**0.211**	0.105	0.105	**0.211**	**0.211**	0.2	0.2
janne_complex	0.842	**0.789**	0.8	0.8	0.158	**0.211**	0.105	0.105	0.158	**0.211**	0.2	0.2
jfdctint	**0.789**	0.895	0.9	1	**0.211**	0.053	0.053	0	**0.211**	0.053	0.1	0
lcdnum	**0.947**	1	1	1	**0.053**	0	0	0	**0.053**	0	0	0
lms	**0.737**	0.842	1	0.9	0.053	**0.158**	0	0.053	0.053	**0.158**	0	0.1
ludcmp	1	**0.947**	1	1	0	**0.053**	0	0	0	**0.053**	0	0
matmult	1	**0.947**	1	1	0	**0.053**	0	0	0	**0.053**	0	0
ns	0.947	**0.895**	0.9	0.9	0.053	**0.105**	0.053	0.053	0.053	**0.105**	0.1	0.1
petrinet	1	1	**0.8**	1	0	0	**0.105**	0	0	0	**0.2**	0
prime	0.842	**0.737**	1	0.8	0.053	**0.105**	0	0.053	0.053	**0.105**	0	0.1
qsort-exam	1	0.947	**0.9**	1	0	**0.053**	**0.053**	0	0	0.053	**0.1**	0
qurt	1	0.895	**0.8**	1	0	**0.053**	**0.053**	0	0	0.053	**0.1**	0
recursion	1	**0.842**	1	1	0	**0.158**	0	0	0	**0.158**	0	0
select	1	1	1	**0.9**	0	0	0	**0.053**	0	0	0	**0.1**
sqrt	**0.895**	**0.895**	1	1	0.053	**0.105**	0	0	0.053	**0.105**	0	0
st	1	**0.789**	1	0.8	0	**0.105**	0	0.053	0	**0.105**	0	0.1
statemate	1	1	1	**0.9**	0	0	0	**0.053**	0	0	0	**0.1**
ud	1	0.947	1	**0.9**	0	**0.053**	0	**0.053**	0	0.053	0	**0.1**

perform relatively the same. Moreover, MO_{eval} using FPA and SPEA was able to find solutions with either better or indifferent *NDR* for 7 and 19 benchmarks, while MO_{approx} using FPA and SPEA found for 6 and 10, respectively. Therefore in terms of *NDR*, MO_{eval}–SPEA contributed to more non-dominated solutions on $\mathcal{P}$ compared to other approaches. Furthermore, MO_{eval} using FPA and SPEA found solutions with either better or indifferent *NDS* for 6 and 13 benchmarks, while MO_{approx} using FPA and SPEA found for 7 and 10, respectively. Therefore in terms of *NDS* metrics, both approaches perform relatively the same.

MO_{approx}–FPA and MO_{eval}–FPA on average found solutions 48.38% and 41.94% times on $\mathcal{P}$, while MO_{approx}–SPEA and MO_{eval}–SPEA on average found solutions 61.3% and 74.19% times on $\mathcal{P}$, respectively. Given that MO_{approx} was able to achieve a 94.12% decrease in the compilation time on average, we can say that the quality of solutions obtained by MO_{approx} is comparable. Therefore, we see a clear merit in using MO_{approx} to perform multi-objective optimization over MO_{eval}.

8 CONCLUSION

In this paper, we proposed a novel approximation model that approximates WCET and energy consumption at compile-time. We evaluated an SPM allocation-based multi-objective optimization problem using two metaheuristic algorithms, i.e., FPA and SPEA. We compared the results from MO_{approx} and MO_{eval} optimization runs to evaluate the efficiency of the proposed approximation model. The approximation errors during the MO_{approx} optimization run were in the acceptable range of 2% - 4% on average. Using MO_{approx}, we achieved on average 94.12% decrease in compilation time compared to MO_{eval}. MO_{approx} and MO_{eval} on average found 54.84% and 58.06% solutions on the final Pareto front $\mathcal{P}$, respectively. Therefore, we can conclusively say that using the proposed approximation model in this paper can drastically reduce

the required compilation time and achieve almost equally good quality Pareto-optimal solutions.

Performing multi-objective optimization at the compiler level requires a large amount of effort in terms of compilation times. Dealing with expensive optimization objectives like WCET and energy consumption contributes immensely to this problem. Therefore, the approximation model proposed in this paper helps to deal with one of the biggest bottlenecks while performing compiler-level multi-objective optimization. The future work is to investigate ways to generalize this approach of approximation toward other compiler-level optimizations.

ACKNOWLEDGMENTS

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.

REFERENCES

[1] AbsInt Angewandte Informatik, GmbH. 2021. aiT Worst-Case Execution Time Analyzers.
[2] Anuradha Balasundaram and Vivekanandan Chenniappan. 2015. Optimal code layout for reducing energy consumption in embedded systems. In *2015 International Conference on Soft-Computing and Networks Security (ICSNS)*. IEEE, 1–5.
[3] Armelle Bonenfant, Denis Claraz, Marianne De Michiel, and Pascal Sotin. 2017. Early WCET prediction using machine learning. In *17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
[4] Heiko Falk, Sebastian Altmeyer, Peter Hellinckx, Björn Lisper, Wolfgang Puffitsch, Christine Rochange, Martin Schoeberl, Rasmus Bo Sørensen, Peter Wägemann, and Simon Wegener. 2016. 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.
[5] Heiko Falk and Jan C Kleinsorge. 2009. Optimal static WCET-aware scratchpad allocation of program code. In *Proceedings of the 46th Annual Design Automation Conference*. 732–737.
[6] Heiko Falk and Paul Lokuciejewski. 2010. A Compiler Framework for the Reduction of Worst-Case Execution Times. *Real-Time Systems* 46, 2 (2010), 251–298.
[7] Heiko Falk, Sascha Plazar, and Henrik Theiling. 2007. Compile-Time Decided Instruction Cache Locking Using Worst-Case Execution Paths. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 143–148. DOI 10.1145/1289816.1289853.
[8] Jan Gustafsson, Adam Betts, Andreas Ermedahl, and Björn Lisper. 2010. 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.
[9] Kenneth Hoste and Lieven Eeckhout. 2008. Cole: compiler optimization level exploration. In *Proceedings of the 6th annual IEEE/ACM International Symposium on Code Generation and Optimization*. ACM, 165–174.
[10] Thomas Huybrechts, Siegfried Mercelis, and Peter Hellinckx. 2018. 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.
[11] Yuriko Ishitobi, Tohru Ishihara, and Hiroto Yasuura. 2007. 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*. IEEE, 13–18.
[12] Shashank Jadhav and Heiko Falk. 2019. Multi-Objective Optimization for the Compiler of Real-Time Systems based on Flower Pollination Algorithm. In *Proceedings of the 22nd International Workshop on Software and Compilers for Embedded Systems*. 45–48.
[13] Andhi Janapsatya, Aleksandar Ignjatović, and Sri Parameswaran. 2006. A novel instruction scratchpad memory optimization method based on concomitance metric. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*. 612–617.
[14] Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 85–103.
[15] Paul Lokuciejewski, Sascha Plazar, Heiko Falk, Peter Marwedel, and Lothar Thiele. 2011. Approximating Pareto optimal compiler optimization sequences—a trade-off between WCET, ACET and code size. *Software: Practice and Experience* 41, 21 (2011), 1437–1458. DOI 10.1002/spe.1079.
[16] Arno Luppold and Heiko Falk. 2017. Schedulability-Aware SPM Allocation for Preemptive Hard Real-Time Systems with Arbitrary Activation Patterns. In *Proceedings of Design, Automation and Test in Europe (DATE)*. Lausanne / Switzerland, 1074–1079.
[17] Kateryna Muts, Arno Luppold, and Heiko Falk. 2018. Multi-Criteria Compiler-Based Optimizations of Hard Real-Time Systems. In *In Proc. of SCOPES* (Sankt Goar, Germany). ACM.
[18] Dominic Oehlert, Arno Luppold, and Heiko Falk. 2016. Practical challenges of ILP-based SPM allocation optimizations. In *Proceedings of the 19th International Workshop on Software and Compilers for Embedded Systems*. ACM, 86–89.
[19] Dominic Oehlert, Arno Luppold, and Heiko Falk. 2017. Bus-aware Static Instruction SPM Allocation for Multicore Hard Real-Time Systems. In *Proceedings of the 29th Euromicro Conference on Real-Time Systems (ECRTS)*. Dubrovnik / Croatia.
[20] Ilya Pavlyukevich. 2007. Lévy flights, non-local search and simulated annealing. *J. Comput. Phys.* 226, 2 (2007), 1830–1844.
[21] Douglas Rodrigues, Xin-She Yang, André Nunes De Souza, and João Paulo Papa. 2015. Binary flower pollination algorithm and its application to feature selection. In *Recent advances in swarm intelligence and evolutionary computation*. Springer, 85–100.
[22] Akash Sachan and Bibhas Ghoshal. 2021. Learning based compilation of embedded applications targeting minimal energy consumption. *Journal of Systems Architecture* 116 (2021), 102116.
[23] YN Srikant and Priti Shankar. 2018. *The compiler design handbook: optimizations and machine code generation*. CRC Press.
[24] Stefan Steinke, Lars Wehmeyer, Bo-Sik Lee, and Peter Marwedel. 2002. Assigning program and data objects to scratchpad for energy reduction. In *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*. IEEE, 409–415.
[25] Vivy Suhendra, Tulika Mitra, Abhik Roychoudhury, and Ting Chen. 2005. WCET centric data allocation to scratchpad memory. In *26th IEEE International Real-Time Systems Symposium (RTSS'05)*. IEEE, 10–pp.
[26] TeamPlay Consortium. 2020. Deliverable D4.5 - Report on Energy Usage Analysis and on Prototype - Version 1.0.
[27] Xin-She Yang, Mehmet Karamanoglu, and Xingshi He. 2014. Flower pollination algorithm: a novel approach for multiobjective optimization. *Engineering Optimization* 46, 9 (2014), 1222–1237.
[28] Eckart Zitzler. 1999. *Evolutionary algorithms for multiobjective optimization: Methods and applications*. Vol. 63. Citeseer.
[29] Eckart Zitzler and Lothar Thiele. 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE transactions on Evolutionary Computation* 3, 4 (1999), 257–271.