# Energy and timing aware synchronous programming

Jiajie Wang, Partha S Roop, Alain Girault

# Energy and timing aware synchronous programming*

JiaJie Wang, Patha S. Roop†
Department of Electrical and Computer Engineering
University of Auckland, New Zealand
jwan232@aucklanduni.ac.nz,
p.roop@auckland.ac.nz

Alain Girault
INRIA
Univ. Grenoble Alpes, France
alain.girault@inria.fr

## ABSTRACT

The synchronous paradigm is widely used for the design of safety critical systems. Such systems, especially in the medical devices domain, must meet strict timing requirements while also ensuring long battery life. As a consequence, they are subject to very strict constraint both regarding their WCRT (Worst-Case Reaction Time) and their WCEC (Worst-Case Energy Consumption, the equivalent constraint for the energy consumption). Many techniques exist to compute an upper bound on the WCRT, but few techniques exist that address both the WCRT and the WCEC. We propose here a static analysis framework where conventional WCRT analysis interacts with a DVFS (Dynamic Voltage Frequency Scaling) algorithm to minimize also the WCEC of the given synchronous program. Our algorithm is able to compute the Pareto front of non-dominated solutions in the (WCRT, WCEC) space. Experimental results reveal that the proposed approach is scalable in terms of analysis time while providing more non-dominated solutions compared to two existing approaches. To the best of our knowledge, the proposed approach is the first to produce energy and timing aware synchronous programs.

## 1. INTRODUCTION

The synchronous paradigm [1] has proved to be very useful in designing safety-critical systems [2, 3]. The advantages are that compilers are grounded on sound semantic foundations based on causality and constructiveness, and hence generate correct-by-construction code that guarantees determinism and reactivity. Determinism ensures that, from any state, given any input vector, at most one reaction is enabled. Reactivity (also called "input receptiveness") ensures that, from any state, given any input vector, at least one reaction is enabled. Moreover, timing verification is supported through a range of static analysis techniques that compute the Worst-Cases Reaction Time (WCRT) [4, 5, 6].

Many safety-critical systems not only require guarantees on functionality and timing but also other non-functional requirements such as battery life [7]. Consider, for example, modern pacemakers, which are implanted even with young adults. Normal battery life for such devices is between 6-10 years and each new implant requires major surgery. In [7], ICD devices[1] and pacemaker failures have also been linked to "premature depletion" of the battery leading to safety issues. This justifies new design methods that consider both timeliness and energy consumption as non-functional requirements during the development phase, in order to provide strong guarantees on both. While synchronous languages offer support for timeliness through static analysis, they offer no support for such bicriteria requirements. Considering this, we propose a design framework, which allows the synchronous paradigm to be used with a commonly available processor technology called Dynamic Voltage and Frequency Scaling (DVFS), which is used for dynamic energy management. We propose a bicriteria heuristic based static analysis algorithm that is designed to work with synchronous programs and offer timeliness guarantees, while also reducing the energy consumption.

### 1.1 Related Work

A processor capable of DVFS may operate at several discrete frequencies. To ensure correct processor operation, each frequency is accompanied by a voltage setting. Thus a processor has several operation modes, each one being a pair (voltage, frequency), denoted $(V, f)$. A point during program execution, where the processor changes its operation mode, is referred to as a *DVFS control point*. DVFS can be controlled by the software to optimize the performance and energy consumption, and find good trade-offs, by changing the operation mode dynamically. For a real-time system, the energy consumption can be reduced by consuming the slack (i.e., the time between task/program completion and the deadline) in exchange for a lower processor frequency.

DVFS algorithms need to make two decisions: the choice of the control point locations in the program and the frequency value associated with each control point. DVFS algorithms may be broadly classified as *online, offline, or hybrid*. Online [8, 9] algorithms decide both the control points and the frequency values during program execution over a

*This work has been partially supported by the RIPPES Inria International Lab.
†Partha Roop acknowledges the German Research Foundation DFG (ME 1427/6-2, HA 4407/6-2) and the research and study leave from Auckland University.

[1]Implantable cardioverter-defibrillator.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Real-Time Operating System (RTOS). They are closely associated with the underlying scheduling policy and hence are scheduler dependent. Offline algorithms [10, 11], on the other hand, make both decision at compile time. Hybrid approaches [12, 13], combine offline and online decisions: DVFS control points are inserted directly into the program statically, but the frequency values of the control points are decided on-line. Both the online and hybrid approaches can be classified as best effort techniques: they rely on extracting the run-time information, such as the actual execution times, in order to compute the next processor frequency. Thus, their effectiveness can only be evaluated using simulation, which may yield variable results depending on the simulation and benchmark settings [14]. Hence, these techniques are unsuitable for safety-critical medical devices, which require strict guarantees.

To ensure that bounds of the system can be computed, we have to consider offline DVFS schemes. These algorithms make compile time decisions regarding both the control point locations and the associated frequency values. The problem may be considered as a bicriteria optimization problem over two antagonistic goals: execution time and energy consumption. For instance, [10, 11] solve this problem by first abstracting the computation of timing. Instead of considering all possible execution paths, they only consider a single execution path derived using profiling. Then, they use Integer Linear Programming (ILP) to optimize the energy consumption, where the execution time of the program is expressed as an ILP constraint. The drawback of such *linearized approaches* is that they may produce under-approximations, which are not desirable in safety-critical systems.

## 1.2 Our approach

We propose, for the first time, a compile time DVFS scheme for synchronous programs. While increasing the frequency reduces the WCRT, such an increase will increase the Worst-Cases Energy Consumption (WCEC). Hence, these two criteria are antagonistic.

To compare tradeoffs in the (WCRT,WCEC) space, we rely on the notion of *dominance* and *Pareto optima*:

- The point $(x, y)$ *weakly dominates* the point $(x', y')$ iff $(x < x' \wedge y = y') \vee (x = x' \wedge y < y')$.
- The point $(x, y)$ *strongly dominates* the point $(x', y')$ iff $(x < x' \wedge y < y')$.
- A point is a *weak Pareto optimum* iff there does not exist another point that strongly dominates it (but there might exist other points that weakly dominate it).
- A point is a *strong Pareto optimum* iff there does not exist another point that (weakly or strongly) dominates it.
- The *Pareto front* is the set of weak and strong Pareto optima.

To avoid potential under-approximations like existing techniques, we do not linearize the problem. Instead, we perform bicriteria minimization using an adaptation of the well-known $\varepsilon$-constraint method [15]. This method is used while performing optimization for two antagonistic criteria such as the WCRT and the WCEC. The $\varepsilon$-constraint method builds the Pareto front incrementally by transforming one of the two criteria into a constraint and minimizing the other criterion under this constraint to obtain one tradeoff. Then, a new value is chosen for the constraint, therefore obtaining a new tradeoff, and so on. At the end, all the dominated tradeoffs are removed to obtain the Pareto front. In our case, we transform WCRT into a constraint, i.e. WCRT $\leq \delta$, where $\delta$ is a given deadline value. We then minimize the WCEC under this constraint. Each value of $\delta$ will potentially produce a different solution in the (WCRT, WCEC) plane. More points are obtained by varying the deadline value by a fixed amount. The developed algorithm is detailed in section 4.

Additionally, we use the four DVFS frequency points of the selected processor to create four points on the Pareto front. This approach is known as the *fixed frequency* approach and it has been established that using fixed frequencies will produce dominant points on the Pareto front [16]. Since these frequencies are far apart, the resulting fixed frequency points are also far apart in the (WCRT,WCEC) plane. The goal of our proposed method is precisely to find more tradeoffs between these fixed-frequency points. Our algorithm, while being sub-optimal, performs no backtracking and hence is very efficient and scalable. Due to sub-optimality, the points obtained may not be guaranteed to be dominant. However, through benchmarking, we show that the algorithm produces dominant points in many benchmarks.

The main contributions of this paper is summarized below:

- We propose the first framework that allows DVFS to be used with the synchronous paradigm for the design of safety-critical systems [7].
- We developed an analysis technique that tackles the bicriteria optimization problem (time, energy) for synchronous programs, without using linearization.
- We have benchmarked our method with real-life synchronous programs. The Pareto optima that we produce dominate the tradeoffs produced by the linearized approach. Moreover, we produce a lot more Pareto optima than the four points produced by the fixed frequency approach, therefore giving the end user more tradeoffs to consider.

The remainder of the paper is organized as follows: We introduce synchronous languages and the target intermediate format in Section 2. Then we formalize the proposed compile time DVFS scheme in Section 3. In Section 4, we present the overview of the proposed framework. Subsequently, the interactive steps of timing analysis and the DVFS algorithm are detailed in Section 5 and Section 6 respectively. We present the evaluation of the proposed approach relative to two existing approaches in Section 7. Finally, Section 8 concludes with discussions on limitations and scope for future work.

# 2. BACKGROUND

## 2.1 Synchronous languages

The foundation of synchronous paradigm is the *synchrony hypothesis*, which assumes that the execution of the program is faster than the environment. Synchronous programs execute in discrete instants, call *ticks*. During each tick, the

environment is sampled at the beginning, then the computations take place, and the outputs are emitted at the end. For the synchrony hypothesis to hold, the execution time of any tick must be shorter than the minimum arrival interval of events, and the analysis to compute the longest execution time of a tick is known as WCRT analysis [4, 5, 6].

## 2.2 Timed Concurrent Control Flow Graph

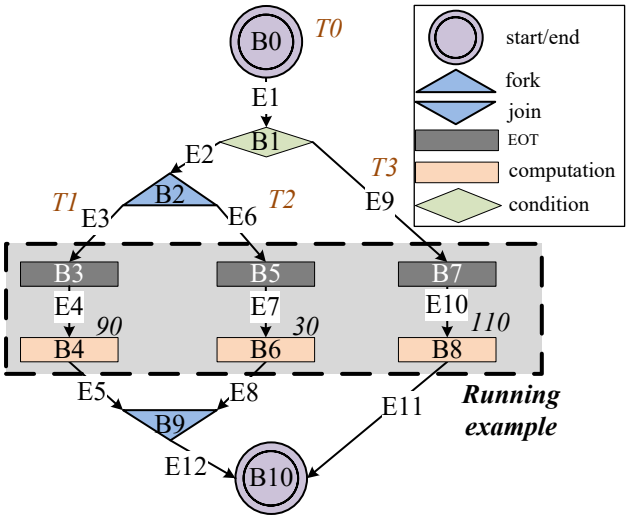


**Figure 1: An example of a TCCFG.**

The technique we propose in this paper is based on the synchronous language PRET-C and its intermediate format Timed Concurrent Control Flow Graph (TCCFG) [17]. TCCFG captures both the high-level control flow of PRET-C programs and the timing information of the underlying platform. As an example, a TCCFG is presented in Fig. 1, which demonstrates a subset of its features. A TCCFG is a directed graph $\{\mathtt{B}, \mathtt{E}\}$ where $\mathtt{B} = \{\mathtt{B0}, \mathtt{B1}, \mathtt{B2}\ldots\}$ is the set of nodes and $\mathtt{E} = \{\mathtt{E0}, \mathtt{E1}, \mathtt{E2}\ldots\}$ is the set of edges. The control flow of a PRET-C program is captured using various notations in TCCFG. The `start/end` nodes (e.g., `B0` and `B10`) mark the beginning and the end of the program. Blocks of sequential computation instructions are encapsulated inside `computation` nodes (e.g., `B4`, `B6` and `B8`), and conditional branches ("if" and "else") are captured by `condition` nodes (e.g., `B1`). Concurrent threads are respectively forked and joined using `fork` and `join` nodes (e.g., `B2` and `B9`). When the parent thread reaches a fork node, it spawns all its child threads and suspends itself, and it resumes when the last child thread terminates (i.e., reaches the join node). The "End of Tick" nodes (EOT) mark the tick/state boundaries of the threads. During each tick, all active threads execute until they have reached their respective EOT nodes, which is therefore a synchronization barrier. When all the active threads have reached their barrier, the program advances to the next tick, and the execution of threads resumes from the EOT nodes where they stopped at the end of the previous tick. It follows that concurrent threads execute in a lock-step manner. PRET-C threads are interleaved only at the tick granularity, i.e, thread switching can only occur after an EOT or a join node.

A fragment of the TCCFG is outlined in Fig. 1 with computation nodes `B4`, `B6`, and `B8`, which will serve as a motivating example to illustrate the proposed framework. Regarding the timing information, each node is annotated with the exact number of processor cycles required to execute it on the target processor. This number is extracted by analyzing the binary code [17]. To have a clear presentation, only the execution costs of `B4`, `B6`, and `B8` are shown in the outlined fragment, which are assumed to be 90, 30, and 110 processor cycles respectively.

## 2.3 Worst-Case Energy Consumption (WCEC)

We want to apply the principle of computing the WCRT as the maximal execution time of all ticks to the energy, with the Worst-Cases Energy Consumption (WCEC). The motivation is to derive, for instance, a lower bound on the life time of a battery powered embedded system. Each PRET-C program will therefore be characterized by its two WCRT and WCEC values. Then, in order to compute several tradeoffs between energy and time, we shall use DVFS, as explained in the next section.

# 3. DVFS SCHEMES FOR SYNCHRONOUS PROGRAMS

The core of the proposed framework is a compile time DVFS framework to produce tradeoffs in the (WCRT,WCEC) plane. Our DVFS framework is tightly coupled with the synchronous semantics: the program has a set of *control points* $\mathcal{C}$, which are the boundaries at which the context switching between threads occurs, i.e., the start, EOT and join nodes. These are ideal places for altering the frequency as they are well marked in synchronous specifications. For example, for the TCCFG in Fig. 1, the set of DVFS control points is $\mathcal{C} = \{\mathtt{B0}, \mathtt{B3}, \mathtt{B5}, \mathtt{B7}, \mathtt{B9}\}$. In addition, we are given a set of pre-defined processor frequencies $\mathcal{F}$. Let the cardinality of $\mathcal{C}$ be $n$ and that of $\mathcal{F}$ be $m$. Then, there are $m^n$ possible alternatives to assign frequency values to the control points. Each such assignment is known as a *DVFS scheme*. The goal of the paper is stated below:

**Goal:** Given a set $\mathcal{C}$ of DVFS control points in a program and a set $\mathcal{F}$ of available frequencies on the processor, a DVFS scheme is a function $\lambda : \mathcal{C} \rightarrow \mathcal{F}$. Given the intermediate TCCFG representation of the program and a timing deadline $\mathcal{T}$, the objective of this paper is to compute a suitable $\lambda$ from the search space, such that the WCRT of the resulting program is less than $\mathcal{T}$ and its WCEC is as small as possible. Then, by varying the deadline $\mathcal{T}$, it is possible to obtain a *set* of tradeoffs, from which all the dominated point (in the Pareto sense, see below) can be removed to obtain the Pareto front.

This is the key challenge of the proposed framework. In Sec 4, we present our static analysis technique which solves this problem by combining an ILP-based WCRT analysis with a greedy heuristic.

## 3.1 A motivating example

In this section, we illustrate how we compute the execution time and energy consumption with a DVFS scheme using the running example outlined in Fig 1. The program starts from the start node `B0`, and executes the condition node `B1`, which has two branches `E2` and `E9`. The branch `E2` leads to the fork node `B2`, which subsequently spawns two threads and eventually leads to the execution of the computation nodes `B4` and `B6`. The alternative branch `E9` leads to

the execution of the computation node B8. In other words, the program either executes B8 or executes B4 and B6 concurrently. The computation nodes B4, B6 and B8 may execute at different frequencies, depending on the values of the DVFS control points at the EOT nodes B3, B5 and B7 respectively. Moreover, since B0 is also a DVFS control point, nodes B1 and B2 should execute at the frequency set in B0. Yet, since there is no computation node between B0 and B3/B5/B7, for the sake of simplicity we do not take this frequency into account in the computations presented in the paper, and similarly for the join node B9. But of course, our tool does take it into account.

**Table 1: The execution time of the computation nodes in Fig. 1 at four different frequencies (Unit: cycle length at 1Mhz).**

| | frequency | | | |
|---|---|---|---|---|
| node | 0.25Mhz | 0.5Mhz | 0.75Mhz | 1Mhz |
| B4 | 360 | 180 | 120 | 90 |
| B6 | 120 | 60 | 40 | 30 |
| B8 | 440 | 220 | 146.7 | 110 |

The execution time of a node varies with the processor frequency. In this paper, we let the execution time of a node executing at the fastest frequency to be equal to its execution cost in processor cycles, and scale the execution time linearly for the slower frequencies. For example, if the running example is executing on a processor that has four frequencies — 0.25Mhz, 0.5Mhz, 0.75Mhz and 1Mhz — then the execution time of the computation nodes B4, B6 and B8 at different frequencies are as in Table 1.

**Table 2: The estimated energy consumption of the computation nodes in Fig. 1 (Unit: Joule/$C \times 10^{12}$).**

| | frequency | | | |
|---|---|---|---|---|
| node | 0.25Mhz | 0.5Mhz | 0.75Mhz | 1Mhz |
| B4 | 5.62 | 22.5 | 50.62 | 90 |
| B6 | 1.87 | 7.5 | 16.87 | 30 |
| B8 | 6.87 | 27.5 | 61.87 | 110 |

The energy consumed by the execution also varies according to the frequency (Table 2). The energy consumption of a node at frequency $f$ is computed by multiplying *its execution cost in processor cycles* with *the energy per cycle at frequency $f$*. The value of the energy per cycle is estimated as follows [18]:

$$\begin{aligned}\text{Energy per cycle at } f &\simeq P_{dynamic} \times \text{cycle length} \\ &\simeq (C \times V^2 \times f) \times (1/f)\end{aligned}$$

where $C$ is the switching capacitance. This is a typical estimation for COMS circuitry, based on the classical model of dynamic power consumption. Additionally, since the voltage can be assumed to be proportional to the frequency (i.e., $V \propto f$ [18]) we may safely assume that the energy per cycle is $\propto f^2$.

## 3.2 Comparing DVFS schemes

Since the number of DVFS schemes grows exponentially, it is important that we determine the relative superiority of one scheme over another. We compare these schemes using their WCRT and WCEC and the notion of Pareto dominance (see Section 1.2). For the running example, the computations of WCRT and WCEC can be abstracted as max operations. Let $t(n)$ denotes execution time of the node $n$ and $e(n)$ denotes its energy consumption. Then, for this running example, the WCRT and WCEC are calculated as:

$$\text{WCRT} = \max(t(B8), (t(B4) + t(B6))) \quad (1)$$

$$\text{WCEC} = \max(e(B8), (e(B4) + e(B6))) \quad (2)$$

**Table 3: DVFS scheme examples.**

| Scheme | DVFS values (MHz) | WCRT | WCEC ($\times 10^{12}$) |
|---|---|---|---|
| 1 | $f_{B4}$=0.5, $f_{B6}$=0.75, $f_{B8}$=0.5 | 220 | 39.37 |
| 2 | $f_{B4}$=0.75, $f_{B6}$=0.5, $f_{B8}$=0.5 | 220 | 58.12 |
| 3 | $f_{B4}$=0.75, $f_{B6}$=1, $f_{B8}$=1 | 150 | 110 |
| 4 | $f_{B4}$=1, $f_{B6}$=0.5, $f_{B8}$=1 | 150 | 110 |

WCRT and WCEC are antagonistic in nature: reducing the WCRT requires to increase the frequencies, which in turn increases the WCEC. Similarly, reducing the WCEC requires to decrease the frequencies, which in turn increases the WCRT. Table 3 shows four of the DVFS schemes for the running example, with their corresponding WCRT and WCEC. The corresponding points are plotted in Fig 2. `Scheme 2` is clearly worse than `scheme 1` as it has the same WCRT but a larger WCEC: in other words, `scheme 1` weakly dominates `scheme 2`. The portion of the plane highlighted in gray contains all the tradeoffs dominated by `scheme 1`. `Scheme 3` and `scheme 4` are distinct in terms of frequency values, but they have identical WCRT and WCEC: this is because we are interested in the worst case values (but they could have distinct average case values for instance). Finally, `scheme 3` and `scheme 1` are incomparable: none dominate the other one. Still, if we are given a strict deadline on the WCRT (as plotted in Fig. 2), then `scheme 1` is the best tradeoff.

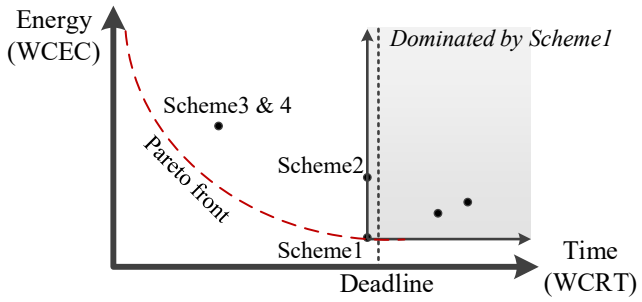


**Figure 2: DVFS schemes for the running example plotted in the (WCRT,WCEC) plane.**

# 4. ALGORITHM FOR WCRT AND DVFS SCHEME COMPUTATION

Given a synchronous program and a timing deadline $\mathcal{T}$, we have to (i) determine the frequency values for the proposed DVFS scheme as defined in Section 3, and to (ii) ensure that WCRT $\leq \mathcal{T}$. To solve this problem, we have developed a framework illustrated in Fig. 3. It consists of two distinct and interactive processes. The frequency values are computed by repeatedly invoking the *timing analysis process* and the *DVFS algorithm* process. Each pro-

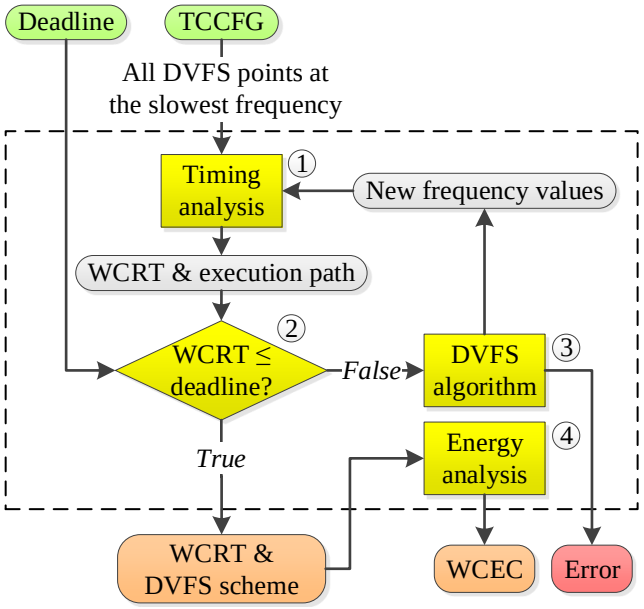


**Figure 3: Overview of the proposed framework.**

cess interacts with the other one until a positive or negative answer is returned. This approach allows us to avoid under-approximation caused by abstracting the program execution.

The developed approach, though iterative, has a wider scope than the recent iterative algorithms [6, 5]. The existing techniques are timing analyses that use Implicit Path Enumeration Technique (IPET) to reduce the search space, therefore improving the scalability for WCRT computation. In our framework, the timing analysis is only one of the internal processes. We extend the classical timing analysis for WCRT with a DVFS analysis to compute DVFS schemes that have a WCRT less than a given deadline and a WCEC that is as small as possible.

Our framework starts with a TCCFG and an initial DVFS scheme where all DVFS control points are set to the slowest frequency (0.25MHz). The analysis first computes the WCRT and its corresponding execution path ①. Then the WCRT is compared to the timing deadline ②. If the WCRT is longer than the deadline, then the DVFS algorithm will take the corresponding execution path of the WCRT, and increase the frequencies of the DVFS control points along that path to reduce its execution time ③. Then the analysis starts over again with the WCRT analysis ①, to search for the other execution paths that exceed the deadline. As soon as the computed WCRT is shorter than the deadline ②, the analysis finishes: the DVFS scheme along with the WCRT and its corresponding execution path are reported. Finally the WCEC of the reported scheme is computed ④. If the DVFS algorithm ③ cannot make an execution path shorter than the deadline (i.e., all DVFS control points are already at the highest frequency), then the analysis terminates with an error, stating that the deadline is not achievable.

In the following sections, we will present more details on the DVFS scheme adjustments ③, which uses a heuristic algorithm to further reduce the analysis time, as well as the analysis technique developed for computing the WCRT ① and the WCEC ④.

# 5. WCRT AND WCEC ANALYSIS

Computing the WCRT and WCEC follow are similar processes as they determine the program path that maximizes a system property, namely the execution time and energy consumption respectively. We propose here a novel ILP-based technique to compute the WCRT and WCEC from a given TCCFG, using execution time and energy cost for respective analysis.

In the following subsections, we will first present the existing ILP formulation for modeling the control flow of a TCCFG [5], then our extension for taking into account DVFS scheme. The TCCFG in Fig. 1 will be used as the running example. For simplicity, the outflow edge of an EOT node or a start node is termed an *EOT edge*.

## 5.1 Modeling the high level control flow

As in IPET [19], the objective function of the formulation is to maximize the sum of all the edges multiplied by their associated costs. For example, the objective function for the example TCCFG is:

$$\max \sum_{i=1}^{N} E_i \times c_i \tag{3}$$

where $N = 12$ in this example, and $c_i$ denotes either the associate cost of $E_i$, which is the execution time in the WCRT analysis, or the energy cost in WCEC analysis.

As for the ILP constraints, two assumptions are made based on the properties of synchronous programs and the nature of ILP. First, the value of each edge is bounded to be either '1' (active) or '0' (inactive). This assumption is valid for any synchronous program as they do not have recursion and instantaneous loops. Second, an edge is set to '1' whenever possible, since the execution/energy costs of all nodes have positive contributions toward the objective function (i.e., all costs are positive values).

### 5.1.1 Computation and condition nodes

The control flow at computation and condition nodes are modeled as inflow edges equals to the outflow edges, as the number of times the control flow enters a computation or condition node must equal to the number of time it leaves the node. For example, the following constraints are created for the running example:

$$\texttt{B1}: E1 = E2 + E9$$
$$\texttt{B4}: E4 = E5$$
$$\texttt{B6}: E7 = E8$$
$$\texttt{B8}: E10 = E11$$

### 5.1.2 EOT nodes

The ILP constraints for EOT nodes are used for emulating the execution of one tick and enforce the parent-child relationships. This is achieved by eliminating the EOT edge combinations, which are unfeasible. For example, the following ILP constraints are generated for the motivating ex-

ample:

$$\texttt{T0-T1}: E1 + E4 \leq 1$$
$$\texttt{T0-T2}: E1 + E7 \leq 1$$
$$\texttt{T0-T3}: E1 + E10 \leq 1$$
$$\texttt{T1-T3}: E4 + E10 \leq 1$$
$$\texttt{T2-T3}: E7 + E10 \leq 1$$

Each ILP constraint consists of a group of EOT edges which are mutually exclusive, i.e., at most one of them can be '1'. For example, the constraint "T0-T1" prevents the threads T0 and T1 to execute concurrently, since T0 is the parent thread of T1. Similarly, the other constraints eliminate other unfeasible combinations. Only T1 and T2 are allowed to execute concurrently.

#### 5.1.3 Fork nodes

Fork nodes spawn child threads when the control flow reaches them. Therefore, all the outflow edges of a fork are active if any of its inflow edge is active. The ILP constraints for the fork node B2 are as follows:

$$\texttt{B2}: E2 = E3, E2 = E6$$

#### 5.1.4 Join nodes

The execution of a join node is denoted by its outflow edge. When the last child thread terminates, the outflow edges of a join node is active; otherwise it should remain inactive. Given that the ILP solver will set the outflow edge to '1' if no ILP constraint is created for the join node (see assumptions at the beginning of Section 5.1), our formulation only needs to model the cases where the outflow edge must be '0'. For example, the following ILP constraints are generated for the join node B9:

$$\texttt{LastChild}: E5 + E8 \geq E12$$
$$\texttt{T1}: (1 - E4 - E2) + (E5) \geq E12$$
$$\texttt{T2}: (1 - E7 - E2) + (E8) \geq E12$$

The outflow edge E12 is on the right hand side of all the ILP constraints. It is '0' if any of the left hand side of the constraints is '0'. Otherwise it is set to '1' by the ILP solver. The LastChild constraint captures the moment when the last child thread reaches the join node B12, during which at least one of E5 and E8 must be active.

The constraints for T1 and T2 capture the states of the two child threads respectively. A child thread forces the outflow edge of the join node to a value '0' if (i) it has an active edge, and (ii) it does not reach the join node. This is captured as two parts in the left hand side of the constraints (separated by parentheses). For example, if any edge in T1 is active, then either its EOT edge E4 is active or the inflow edge of the fork node E2 is active, since all the edges in T1 are initiated from them. So the first part "$1-E4-E2$" is '0' if this is true ($E4$ and $E2$ are mutually exclusive from each other because of the constraints for EOT nodes). The second part "$E5$" is '0' if the control does not reach the join node. If any of the two parts is '1', then the outflow edge $E12$ is set to '1' provided that T2 also meets the same condition.

### 5.2 Modeling the DVFS scheme

Processor cycle has been the primary unit in conventional timing analysis. The actual physical time is calculated by multiplying the number of clock cycles with the cycle length (which is $1/f$). However, when DVFS is employed, the processor frequency changes during the execution, and this effect must be taken into consideration in the timing analysis. This changes of frequency complicates the ILP formulation as it introduces multiple costs for each node.

Conventional IPET [19] cannot model nodes with variable costs. To cope with this limitation, we consider a TCCFG with DVFS as multiple virtual TCCFGs, where each TCCFG corresponds to one specific frequency in $\mathcal{F}$. An illustration of this for our example TCCFG is shown in Fig. 4. The frequency switching is emulated as the control flow jumping between these virtual TCCFGs.

The virtual TCCFGs are identical to the original, except that the cost of each node is based on the associated frequency. The calculations of execution and energy cost are shown in Section 3.1. The switching penalty is modeled as a constant cost and is added to the start node, and all the EOT and join nodes (i.e., all the nodes in $\mathcal{C}$).

Now, the virtual TCCFGs introduce multiple avatars for each edge in the original TCCFG. For example, with processor frequencies 0.25MHz, 0.5MHz, 0.75MHz and 1MHz, an edge $E_i$ in the original TCCFG becomes four avatars, one for each frequency. We differentiate these avatars using suffixes, so that the avatars of Ei are Ei_a, Ei_b, Ei_c and Ei_d, corresponding to 0.25MHz, 0.5MHz, 0.75MHz and 1MHz respectively. Each avatar, like the original edge, is a binary value.

At any time, only one avatar of an edge can be active in an execution path, since a node can only execute at one frequency. Thus the sum of all the avatars of an edge should be at most '1'. To enforce this, the following ILP constraint is created for each edge:

$$E_i\_a + E_i\_b + E_i\_c + E_i\_d \leq 1$$

After creating all the avatars, we rebuild all the ILP constraints for a TCCFG with these avatars. We developed a systematic approach, which can preserve the modeling of the high level control flow while integrating the DVFS scheme. Each ILP formulation is treated differently depending whether the node has a DVFS control point or not.

### 5.3 Nodes without DVFS control points

For the computation nodes, condition nodes and fork nodes, which do not have frequency control points, we consider each virtual TCCFG independently, and generate ILP constraints for each using the corresponding avatars. For example, the following ILP constraints are generated for the running example:

0.25MHz TCCFG:

$$\texttt{B1}: E1\_a = E2\_a + E9\_a$$
$$\texttt{B2}: E2\_a = E3\_a; E2\_a = E6\_a;$$
$$\texttt{B4}: E4\_a = E5\_a$$

0.5 MHz TCCFG:

$$\texttt{B1}: E1\_b = E2\_b + E9\_b$$
$$\texttt{B2}: E2\_b = E3\_b; E2\_b = E6\_b;$$
$$\texttt{B4}: E4\_b = E5\_b$$
$$\ldots$$

The modeling of the control flow is preserved, as the generated ILP constraints are identical to the original with respect

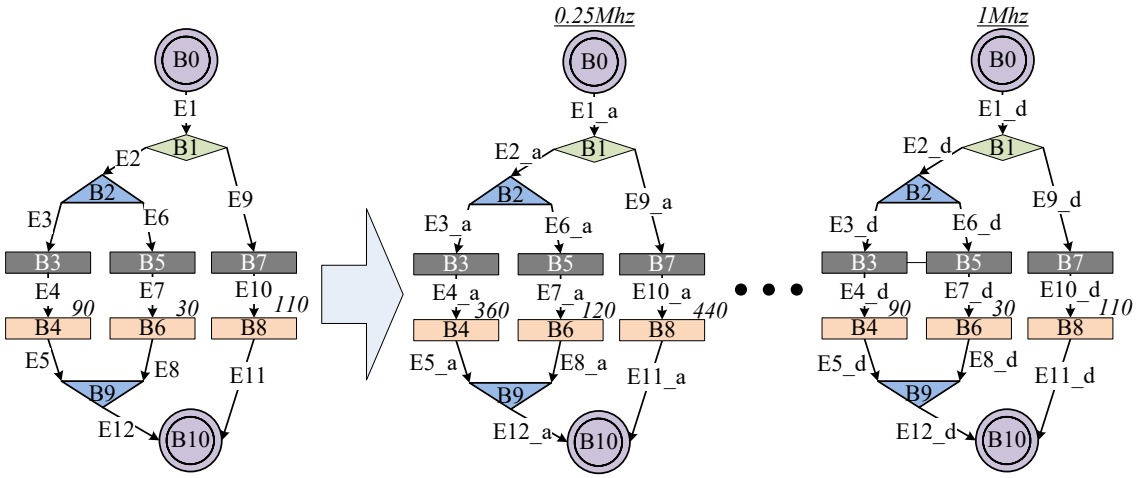


**Figure 4: Converting the TCCFG in Fig 1 into four virtual TCCFGs for emulating DVFS.**

to each virtual TCCFG. However, the processor frequency cannot be changed during the execution of these nodes. For example, if the condition node `B1` executes at 0.5MHz, then its inflow edge `E1_b` is '1' (the other avatars are '0'). The ILP constraint will allow `B1` to choose its outflow edge, but the processor frequency must remain to be 0.5MHz (i.e., the only choices are `E2_b` and `E9_b`).

## 5.4 Nodes with DVFS control points

For the EOT and join nodes that are in $\mathcal{C}$, we consider the all the virtual TCCFGs as a whole, and create a combined set of ILP constraints to allow the control flow to jump between them. We replace each edge in the original ILP formulation with the sum of its avatars. For example, the following ILP constraints are generated for our running example:

EOT nodes:

`T0-T1`: $(E1\_a + E1\_b + E1\_c + E1\_d) +$
$(E4\_a + E4\_b + E4\_c + E4\_d) \leq 1$

`T0-T2`: $(E1\_a + E1\_b + E1\_c + E1\_d) +$
$(E7\_a + E7\_b + E7\_c + E7\_d) \leq 1$

...

Join node `B9`:

`LastChild`: $(E5\_a + E5\_b + E5\_c + E5\_d) +$
$(E8\_a + E8\_b + E8\_c + E8\_d) \geq$
$(E12\_a + E12\_b + E12\_c + E12\_d)$

`T1`: $[1 - (E4\_a + E4\_b + E4\_c + E4\_d) +$
$- (E2\_a + E2\_b + E2\_c + E2\_d)] +$
$(E5\_a + E5\_b + E5\_c + E5\_d) \geq$
$(E12\_a + E12\_b + E12\_c + E12\_d)$

`T2`: $[1 - (E7\_a + E7\_b + E7\_c + E7\_d) +$
$- (E2\_a + E2\_b + E2\_c + E2\_d)] +$
$(E8\_a + E8\_b + E8\_c + E8\_d) \geq$
$(E12\_a + E12\_b + E12\_c + E12\_d)$

These ILP constraints preserve the modeling of the control flow while allowing the frequency to be changed. For example, if `T1` executes at 0.25MHz (i.e., `E4_a` and `E5_a` are '1') and `T2` executes at 0.5MHz (i.e., `E7_b` and `E8_b` are '1'), then the control flow of the two threads are on different virtual TCCFGs, but the ILP constraints still allow their join node `B9` to be active. Furthermore, the join node `B9` can switch to any frequency (i.e., it is free to select any avatars of `E12`).

## 5.5 Choosing a frequency

Initially, the ILP solver is free to choose any frequency for the control points. As the analysis proceeds, the DVFS algorithm will select specific frequencies for the control point to reduce the execution time. This selection is modeled differently for the WCRT and WCEC analyses. The ILP solver always prioritizes the frequency that maximizes the objective function: $\texttt{E}_\texttt{i}\texttt{\_a} > \texttt{E}_\texttt{i}\texttt{\_b} > \texttt{E}_\texttt{i}\texttt{\_c} > \texttt{E}_\texttt{i}\texttt{\_d}$ in the WCRT analysis, and in reverse order in the WCEC analysis. Therefore, we can choose a specific frequency by setting the unwanted ones to '0', which forces the solver to take the next best. For example, to choose 0.5MHz for the EOT node `B3` (i.e., `E4_b` to be '1'), we create the following ILP constraints respectively for the WCRT and WCEC analyses:

| In WCRT analysis: | In WCEC analysis: |
|---|---|
| $E4\_a = 0$ | $E4\_d = 0$ |
| | $E4\_c = 0$ |

# 6. THE DVFS ALGORITHM

We propose a greedy heuristic for the DVFS algorithm for adjusting the frequency of the DVFS control points along an execution path. The idea of the algorithm is to reduce the execution time of the path as much as possible during each frequency adjustment. To do so, the algorithm only increases the frequency values (i.e., frequency values are never decreased). It adjusts one DVFS control point with one frequency step at a time and it prioritizes the DVFS control points which yield the largest reduction in execution time.

For example, let us consider an execution path from the running example in Fig. 1, whose execution time is the sum of the execution time of `B4` and `B6`. Let us assume both nodes are executing at 0.25MHz, which results in an execution time equal to 360+120. The algorithm will select `B3`

and increase its frequency by one step, to 0.5MHz (execution time: 180+120). Indeed, this adjustment yields the largest reduction in execution time compared with increasing the frequency of `B5` by one step (execution time: 360+60). The execution time of the whole execution path is re-calculated with each frequency adjustment, and the process repeats until the execution time of the path is shorter than the deadline (or a failure is reported if it is not possible). This heuristic algorithm effectively reduces the search space.

# 7. RESULTS

Recall that for a multi-criteria optimization problem, there is no single optimal solution. In this section, we compute the estimated Pareto front for a set of benchmark programs using the proposed approach, and compare the results with two existing approaches: the fixed frequency approach, which uses a single processor frequency throughout the execution, and the linearized approach [10, 11]. We first discuss the results in terms of WCRT and WCEC, followed by an comparison in analysis time.

## 7.1 Benchmark method

The target platform uses a Microblaze processor with four frequencies: 1Mhz, 0.75Mhz, 0.5Mhz, and 0.25Mhz. The frequency switching penalty is assumed to be five processor cycles at 1Mhz.

**Table 4: Information of the benchmark programs.**

| Name | LOC | Threads | DVFS ctrl points |
|---|---|---|---|
| Channel Protocol | 591 | 7 | 20 |
| Flasher | 816 | 7 | 24 |
| Robot Sonar | 962 | 7 | 18 |
| Cruise Controller | 2302 | 25 | 60 |

Our benchmark programs are industrial applications taken from [20]. For each benchmark, we build the Concurrent Control Flow Graph (CCFG), then we compile it into C code and then into binary to obtain the exact number of processor cycles of each node, therefore resulting in the TCCFG (that is, the Timed CCFG). The benchmarks are summarized in Table 4.

We applied the three approaches to each of the benchmark programs. For our bicriteria algorithm, the initial deadline is selected to be the WCRT of the program executing at 1MHz (i.e., the tightest deadline that the system can meet, where all DVFS points are set to the fastest frequency). The deadline increases by 20% of the starting value each time until the deadline is equal to the WCRT of the program executing at 0.25MHz (i.e., all DVFS points are set to the slowest frequency).

## 7.2 Computing the Pareto front

The fixed frequency approach serves as a baseline for comparison, which represents the case of executing a conventional synchronous program at a fixed frequency, without using DVFS. There is no frequency switching penalty, and the WCRT and WCEC are computed using the ILP technique presented in Section 5. The fixed frequency approach is benchmarked for each of the four available frequencies for each program. The results are plotted in the (WCRT,WCEC) plane in Fig. 5 where each blue triangle represents one such fixed frequency tradeoff. All these four points are strong Pareto points, meaning that there does not exist any point better in both criteria WCRT and WCEC. In Fig. 5, we also outline the portion of the (WCRT,WCEC) plane dominated by each fixed frequency point.

The linearized approach handles the bicriteria optimization problem by first linearizing the program execution path using profiling [10, 11]. To implement this principle, we use the critical path obtained through conventional WCRT analysis as the profiled path, and we optimize the energy over that path. We also restrict the allocation of DVFS control points to be at the start, EOT, and join nodes to be consistent with the proposed approach. The results are plotted in Fig. 5 as green circles. Most of the schemes produced by the linearized approach overlaps with each other, thus only three points are visible in each graph. All of these points are dominated by the points produced by the other approaches. More importantly, due to the abstraction, the linearized approach misses the given deadline most of the time.

Finally, for each benchmark program, we produce tradeoffs with our bicriteria approach by varying the deadline (Section 7.1). The results are plotted in Fig. 5 as red squares.

Compared with the fixed frequency approach, our approach is disadvantaged due to the frequency switching penalties. However, we produce a lot more tradeoffs. For instance, for the Channel Protocol (Fig. 5a), the Pareto front consists of 12 points: 4 are fixed frequency while 8 are obtained with our approach (recall that the Pareto front consists of all the points that are non-dominated). This gives a lot more choices to the end user. Consider again the Channel Protocol and suppose that the user wishes to find the best DVFS scheme such that the WCRT be less than 3500 clock cycles, where "best DVFS scheme" means "the DVFS scheme with the smallest WCEC" because the WCRT aspect is already taken care off through the 3500 clock cycles constraint. As shown in Fig. 5a, the tradeoffs with the smallest WCEC found are:

$$\begin{aligned} \text{linearized method: } & \text{WCEC} = 766 \\ \text{fixed frequency method: } & \text{WCEC} = 256 \\ \text{our approach: } & \text{WCEC} = 176 \end{aligned}$$

Of course, the ranking between the three approaches will depend on the deadline chosen by the user, but it is obviously better to have more points in the Pareto front, which demonstrates the usefulness of our approach.

Finally, in some cases our approach fails to provide non-dominated solutions in key portions of the search space. This is the case of the Flasher and the Cruise Controller, where our approach failed to provide non-dominated solutions between the two fixed frequency Pareto optima corresponding respectively to 0.5MHz and 0.75MHz (see Figs. 5b and 5d). A future work direction will be to investigate how to improve the DVFS schemes found by our algorithm by a local search in the search space.

## 7.3 Analysis time

Analysis time is an important quantitative consideration. Off-line DVFS algorithms are hard to scale due to the large search space [13]. The analysis time for the largest benchmark program, the Cruise Controller, is recorded in Table 5. All three approaches are practical to use. The fixed frequency approach does not require any analysis, and both the proposed approach and linearized approach finish the analysis in matter of seconds.

The linearized approach has a constant analysis time as its

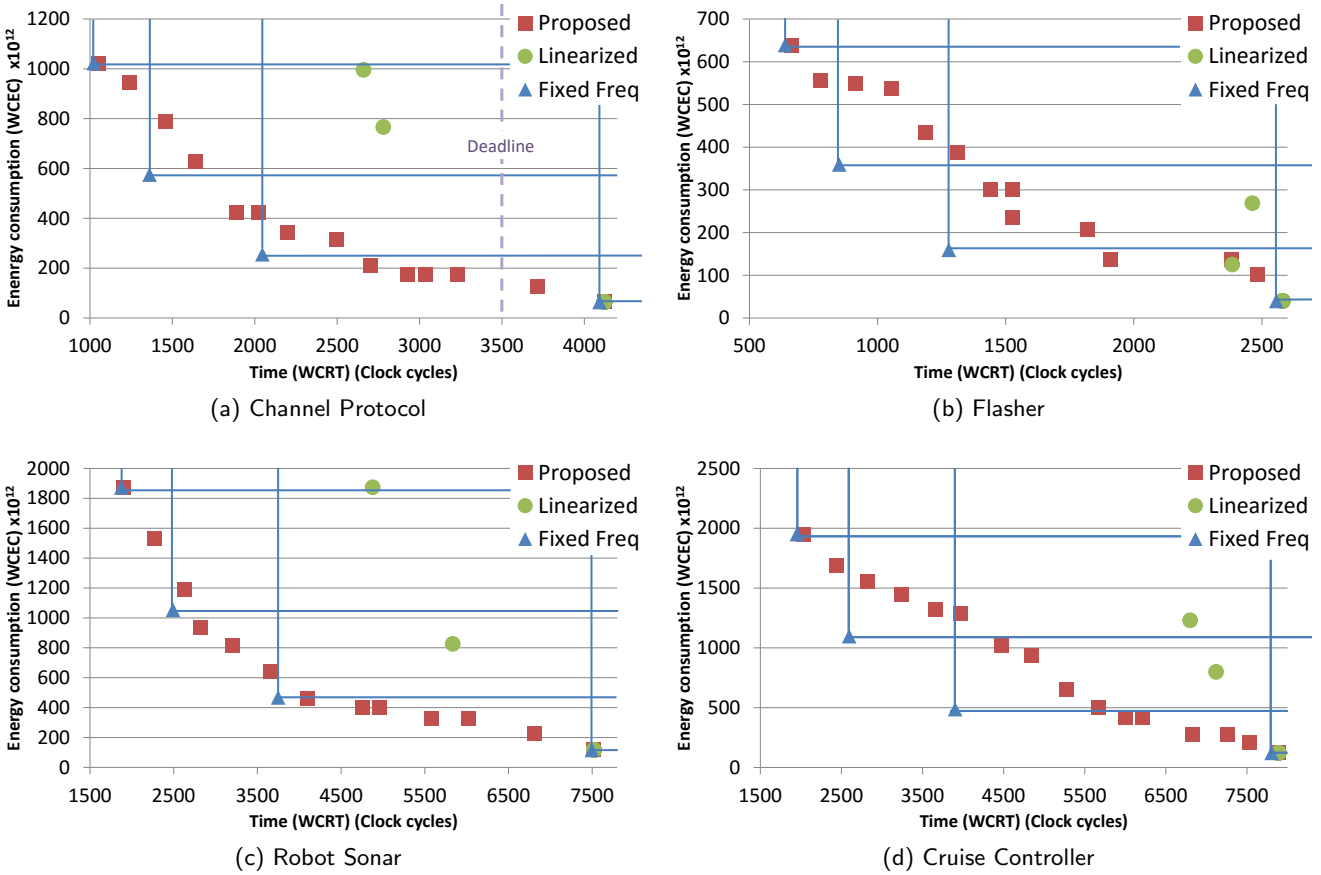


**Figure 5: Benchmark results. Comparing three approaches in the (WCRT,WCEC) plane).**

search space does not change. On the other hand, the analysis time of the proposed approach increases as the deadline get tighter. This is because the starting point of our algorithm is the DVFS scheme where all control points are set to the smallest frequency (0.25MHz), so a lot of iterations must be performed before reaching a DVFS scheme that meets a tight deadline. A future work direction will be to investigate whether it is doable to initiate our algorithm with another fixed frequency scheme, e.g., 0.75MHz for deadlines less than 2500 clock cycles in the case of the Robot Sonar.

However, given the fact that the search space of the proposed approach is much larger than the linearized approach, it is nice to observe that they have comparable analysis time. We conclude that the greedy approach that performs no backtracking for computing a DVFS scheme ($\lambda$) significantly reduces the analysis time.

## 8. CONCLUSIONS AND FUTURE WORK

Synchronous languages provide efficient programming of embedded safety critical systems. Recent results have focused on computing their Worst-Cases Reaction Time (WCRT) with ILP based methods improved with IPET [4, 5, 6]. This is perfectly relevant to validate whether a given system meets its timing constraint. However, modern embedded systems must also satisfy strict *energy consumption constraints*, especially for tiny devices such as sensor networks, biomedical implants, and ICD devices. Indeed, premature battery wear-off is a life threatening issue in such devices [7]. Regarding synchronous programs, the energy equivalent of the WCRT is the Worst-Cases Energy Consumption (WCEC), which is a guaranteed bound of the energy consumption of the system during one reaction.

We have proposed an original bicriteria (WCRT,WCEC) static analysis optimization method to tackle this. It combines a classical WCRT analysis with an iterative DVFS algorithm for synchronous programs. The proposed approach provides a lot more (WCRT,WCEC) tradeoffs than the previous techniques. Because the two criteria are antagonistic, we advocate that our algorithm must provide as many as possible non-dominated points (in the Pareto dominance sense) in the (WCRT,WCEC) space. To achieve this, we transform the bicriteria optimization problem into a single criterion optimization problem by turning the WCRT criterion into a constraint and by minimizing the WCEC under this constraint. By varying the WCRT constraint as in the $\varepsilon$-constraint method [15], we are able to produce a whole set of non-dominated Pareto optima, that is, the Pareto front. To the best of our knowledge, this is the first bicriteria (WCRT,WCEC) optimization method for synchronous programs.

The originality of our approach is to use a new iterative algorithm that explores the search space incrementally. Thanks to benchmarking with real-life programs, we have proved that our algorithm can produce a lot more points than the fixed frequency and the linearized approaches [10, 11]. The benchmarks also show that, even though the search space is exponential, the analysis time required by our algorithm is linear, thanks to a fast heuristic search.

**Table 5: Analysis time comparison between the proposed approach and the linearized approach.**

| Multiplier | Analysis Time (s) | |
|---|---|---|
| | Proposed | Linearized (Excluding profiling and WCRT analysis) |
| 1.0 | 3.03 | 1.04 |
| 1.2 | 2.19 | 1.15 |
| 1.4 | 2.25 | 1.12 |
| 1.6 | 2.59 | 1.10 |
| 1.8 | 2.39 | 1.14 |
| 2.0 | 3.11 | 1.08 |
| 2.2 | 2.54 | 1.14 |
| 2.4 | 2.83 | 1.10 |
| 2.6 | 3.12 | 1.15 |
| 2.8 | 2.27 | 1.15 |
| 3.0 | 2.00 | 1.11 |
| 3.2 | 2.50 | 1.18 |
| 3.4 | 1.58 | 1.16 |
| 3.6 | 1.36 | 1.23 |
| 3.8 | 0.45 | 1.24 |
| 4.0 | 0.19 | 1.11 |

Regarding future work, we plan to investigate several avenues. First, we wish to improve the DVFS schemes found by our algorithm by a local search in the state space, for instance by lowering the frequencies in the control paths that are not the critical path. Second, we wish to accelerate our heuristic by segmenting the search space into intervals bounded by fixed frequencies, and by starting our algorithm with the fixed frequency scheme corresponding to the targeted WCRT deadline.

# 9. REFERENCES


[1] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. De Simone, "The synchronous languages 12 years later," *IEEE*, vol. 91, no. 1, pp. 64–83, 2003.

[2] F. X. Dormoy, "SCADE 6: a model based solution for safety critical software development," in *Proceedings of European Congress on Embedded Real Time Software (ERTS)*, 2008, pp. 1–9.

[3] M. Huhn and S. Bessling, "Towards certifiable software for medical devices: The pacemaker case study revisited," in *Proceedings of International Workshop on Harnessing Theories for Tool Support in Software (TTSS)*, 2011, pp. 8–14.

[4] M. Kuo, R. Sinha, and P. S. Roop, "Efficient WCRT analysis of synchronous programs using reachability," in *Proceedings of the Design Automation Conference (DAC)*, 2011, pp. 480–485.

[5] J. J. Wang, P. S. Roop, and S. Andalam, "ILPc: A novel approach for scalable timing analysis of synchronous programs," in *Proceedings of the international conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, 2013, pp. 1–10.

[6] P. Raymond, C. Maiza, C. Parent-Vigouroux, F. Carrier, and M. Asavoae, "Timing analysis enhancement for synchronous program," *Real-Time Systems*, vol. 51, no. 2, pp. 192–220, 2015.

[7] H. Alemzadeh, R. K. Iyer, Z. Kalbarczyk, and J. Raman, "Analysis of safety-critical computer failures in medical devices," *IEEE Security & Privacy*, vol. 11, no. 4, pp. 14–26, 2013.

[8] H. Aydin, V. Devadas, and D. Zhu, "System-Level Energy Management for Periodic Real-Time Tasks," in *Proceedings of Real-Time Systems Symposium (RTSS)*, 2006, pp. 313–322.

[9] N. AbouGhazaleh, D. Mossé, B. R. Childers, and R. Melhem, "Collaborative Operating System and Compiler Power Management for Real-time Applications," *ACM Transaction on Embedded Computing (TECS)*, vol. 5, no. 1, pp. 82–115, 2006.

[10] F. Xie, M. Martonosi, and S. Malik, "Compile-time Dynamic Voltage Scaling Settings: Opportunities and Limits," in *Proceedings of Programming Language Design and Implementation (PLDI)*, 2003, pp. 49–62.

[11] P.-K. Huang and S. Ghiasi, "Leakage-aware intraprogram voltage scaling for embedded processors," in *Proceedings of the Design Automation Conference (DAC)*, 2006, pp. 364–369.

[12] J. Seo, T. Kim, and K.-S. Chung, "Profile-based optimal intra-task voltage scheduling for hard real-time applications," in *Proceedings of the Design Automation Conference (DAC)*. New York, NY, USA: ACM, 2004, pp. 87–92.

[13] P.-K. Huang and S. Ghiasi, "Efficient and Scalable Compiler-directed Energy Optimization for Realtime Applications," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 12, no. 3, pp. 27:1–27:16, 2008.

[14] W. Kim, D. Shin, H.-S. Yun, J. Kim, and S. L. Min, "Performance comparison of dynamic voltage scaling algorithms for hard real-time systems," in *Proceedings of Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2002, pp. 219–228.

[15] Y. Haimes, L. Lasdon, and D. Wismer, "On a bicriterion formulation of the problems of integrated system identification and system optimization," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 1, pp. 296–297, 1971.

[16] N. B. Rizvandi, J. Taheri, and A. Y. Zomaya, "Some observations on optimal frequency selection in DVFS-based energy consumption minimization," *Journal of Parallel and Distributed Computing*, vol. 71, no. 8, pp. 1154–1164, 2011.

[17] S. Andalam, P. S. Roop, A. Girault, and C. Traulsen, "A Predictable Framework for Safety-Critical Embedded Systems," *IEEE Transactions on Computers (TC)*, vol. 63, no. 7, pp. 1600–1612, Jul. 2014.

[18] T. D. Burd and R. W. Brodersen, "Energy efficient CMOS microprocessor design," in *Proceedings of Hawaii International Conference on System Sciences (HICSS)*, vol. 1, 1995, pp. 288–297.

[19] P. Raymond, "A general approach for expressing infeasibility in implicit path enumeration technique," in *Proceedings of the International Conference on Embedded Software (EMSOFT)*, 2014, pp. 1–9.

[20] L. H. Yoong, P. S. Roop, and Z. Salcic, "Implementing constrained cyber-physical systems with IEC 61499," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12S, no. 1, 2013.