A Nonlinear Programming Based Framework for Energy Optimization in Real-time Systems

Abstract—**In the energy minimization of real-time systems, the designers often face a challenging problem in which the schedulability conditions may be nonlinear, non-continuous functions of the decision variables, and the objective function or other constraints may also be nonlinear and even non-convex. In this paper, we propose a general framework for such optimization problems, by exploiting nonlinear programming and its gradient-based solution methods. To handle the non-continuous schedulability constraints, we introduce several new adaptations to improve overall performance. The proposed method does not impose any restrictions on the schedulability analysis technique, the objective function, or any additional constraints, hence is very general and compatible with a broad range of real-time systems. Experiments demonstrate $5000\times$ speedup for the proposed approach compared with state-of-art methods while maintaining very similar solution quality.**

I. Introduction

The design of real-time systems should satisfy the real-time schedulability constraints, but they are often subject to many other requirements and objectives, including limited resources (e.g., memory), cost, quality of control, power, and energy consumption. For example, many application domains of real-time systems face the stringent and often conflicting constraints on size, weight, power, and cost (SWaP-C), such as drones [1], medical devices [2], and internet of things [3]. In these application domains, it is important to perform optimization in order to find the best setting (i.e., optimized according to an objective function) while satisfying all the critical requirements.

Formally, an optimization problem is defined by decision variables, constraints, and an objective function. The *decision variables* represent the set of design choices under the designers' control. The set of *constraints* forms the feasibility region, the domain of the allowed values for the decision variables. The *objective function* characterizes the optimization goal. In general, the optimal design can be obtained by solving an optimization problem where the objective function is optimized within the feasibility region. For real-time systems, the *feasibility region* (also called *schedulability region* if concerning only real-time schedulability) must only contain the designs that satisfy the *schedulability constraints* whereby tasks complete before their deadlines.

We use energy as an example of design considerations beyond real-time schedulability. Battery-powered real-time systems usually can only run for a limited time without being recharged. As such, more efficient energy management strategy not only increases the usage time of both applications and battery life, but also brings better user performance to most customers. In this paper, we mainly consider one of the most popular optimization strategies used in real-time embedded systems: Dynamic Voltage and Frequency Scaling (DVFS) [4]. By sacrificing part of the response time and running tasks at a lower frequency, DVFS is able to save dynamic power consumption to a big extent because power consumption usually follows a quadratic or cubic relationship with run-time frequency,

In this paper, we consider the energy optimization for real-time systems. Although our approach applies to a broad range of scheduling models and schedulability analysis techniques, it is best suited for systems scheduled with fixed priority. Handling the schedulability constraints in optimizing such systems is very challenging. Even for the basic Liu-Layland task model, it has been demonstrated that the schedulability analysis is NP-hard [5]. The schedulability region is shown to be non-convex with respect to the worst-case execution times [6].

Hence, one of the major limitations in the existing approaches lies in their computation complexity. Most algorithms have exponential complexity in the worst case [7], [8], and such big overhead limits their application, especially in online situations, where major energy saving could happen because many pessimistic assumptions must be added in offline analysis to guarantee schedulability. Besides, most of these methods are proposed aiming at a specific type of systems, which limits their application further.

In this paper, we propose a very general DVFS optimization framework for FTP scheduling algorithms. By formulating it as a non-linear least square problem and solving it with gradient-based methods, efficient solutions can be explored from special problem structures such as sparsity [9]. The application of the proposed method is not limited to a specific type of schedulability analysis such as utilization test in EDF, but works with most types of systems. The computation cost is dominated by initial schedulability analysis. Although it usually has pseudo-polynomial complexity, it only has to be analyzed for one time during the optimization process. Apart from it, the optimization part usually only requires $O(N^2)$ or even $O(N)$ during most iterations, where N is the number of tasks.

We also performed extensive experiments for evaluation. The proposed algorithm runs 1000 times faster than state-of-art algorithms, while maintaining performance that is usually close to global optimal results generated by brute-force methods.

II. Related Work

Early research of energy management based on DVFS was explored by Chen *et al.* [10] for simple task systems. After it, more DVFS algorithms are proposed to work with EDF scheduler [11]–[15], Rate Monotonic scheduler [7], [11]. Most of related work focus on scheduling systems with EDF scheduler because of the easy schedulability analysis, while those optimizing energy for FTP schedulers usually have large computation cost.

Within DVFS, Bambagini *et al.* [4] categories based on system type (single-core (most of the early work falls in this category) versus multi-core processors [16]–[18]) and type of slack that algorithms reclaim for (static slack [19], dynamic slack [15], [20], or mixture of both [21]). Basically, the dynamic slack algorithms try to make use of the difference of real execution time and worst-case execution time, and improve the potential performance. Mostly, algorithms mentioned above are usually specified for one type of system or one type of mechanism. However, the optimization framework introduced in this paper can be easily adapted to optimize under situations if only appropriate schedulability analysis are provided.

Many different optimization methods are adopted in the real-time literature. According to categories given by Zhao *et al.* [8], popular methods include heuristics such as simulated annealing [22], genetic algorithms [23], existing classical optimization frameworks such as branch-and-bound [24], mixed integer linear programming [25], convex programming [13], or some customized optimization methods for specific situations, such as priority assignment [26]. Many of these methods except convex optimization have large computation cost and also cannot provide performance guarantee, while convex optimization is usually very hard to be directly applied in real-time systems because the schedulability region is usually not convex. Provided great and inspired observation, the last category can be seen as a kind of art and the performance could be very good. However, it is usually limited to a specific type of problems. In contrast of the previous methods, the nonlinear programming framework proposed in this paper is able to handle a large variety of systems with only the necessary information provided (such as providing schedulability analysis). By exploiting the problem structure, which exists for most response time analysis or demand bound constraints, the overall computation cost could be very low while still delivering excellent results.

Comparing with related work, this paper is novel in the following aspects:

- This paper proposes a novel optimization framework for DVFS. Based on classical nonlinear optimization method, this paper models the DVFS as a least-square problem, and use gradient-based methods to solve it iteratively.
- The proposed method is not limited to specific types of schedulability analysis. Actually, any schedulability analysis can be considered even if schedulability follows a non-sustainable or nonlinear relationship with respect to computation time variables. To the best knowledge of authors, this type of problems are not well explored in the current literature.
- The proposed algorithm is able to run extremely fast compared with state-of-art algorithms, and generate excellent results close to global optimal in random task sets.

III. System model

1) Task system: Without loss of generality, we use the following task system model to illustrate our optimization approach, the application of the proposed method is actually far beyond the illustration examples.

There are N periodic, preemptive tasks that run on a uniprocessor system. Each task τ_i is characterized by a period T_i, initial worst-case execution time (WCET) $c_i^{(o)}$, a constrained deadline D_i, and a fixed priority P_i. The hyper-period (least common multiple of task periods) of the task sets is denoted as H. The task systems are scheduled based on priorities, and high priority tasks always preempt low priority tasks when they become available. $hp(i)$ denotes the set of tasks with higher priority than τ_i, and $hep(i)$ denotes the set of tasks with higher or equal priority than τ_i. Without loss of generality, we let τ_0 have the highest priority, and τ_{N-1} have the lowest priority. The response time can be described easily as follows:

$$r_i = c_i^{(o)} + \sum_{j \in hp(i)} \left\lceil \frac{r_i}{T_j} \right\rceil c_j \tag{1}$$

The task set is schedulable if

$$\forall \tau_i, r_i \leq d_i \tag{2}$$

2) Energy model: We adopt a simple, but widely used energy model in the literature [4], [8] in this paper. If we denote the base processor run-time frequency as 1, given a run-time frequency f, τ_i's execution time requirement becomes

$$c_i = \frac{c_i^{(o)}}{f} \tag{3}$$

The energy consumption power is usually assumed to follow a polynomial relationship with run-time frequency f. One commonly used model is

$$P(f) = \alpha f^3 \tag{4}$$

where α is a circuit-dependent coefficient. Other types of energy models such as adding static component, changing the order in Equation 4 can also be used in the following optimization procedure.

Correspondingly, the energy consumption within a hyper-period H is

$$E_i = \frac{H}{T_i} c_i \alpha (\frac{c_i^{(0)}}{c_i})^3 \tag{5}$$

A. Problem formulation

We consider a general energy optimization problem subject to schedulability constraints in a hard real-time system. To be more specific, we hope to solve the following problem:

$$\min_{\mathbf{c}} \sum_i \frac{H}{T_i} c_i \alpha (\frac{c_i^{(0)}}{c_i})^3 \tag{6}$$

$$\forall i, r_i \leq d_i \tag{7}$$

The response time analysis is usually assumed to be given, such as Equation 1. Since H is usually a constant number during optimization, we will leave it out in the future.

Although the objective function 6 has a convex form with respect to variables $\mathbf{c}$, the real-time schedulability constraints are usually much more complicated. Even as simple as the example task model, such constraints are usually discrete and nonlinear. Besides, another major issue in solving such problems lies in the computation cost. It is known that response time estimation with similar form as Equation 1 has a pseudo-polynomial complexity [27]. As such, performing schedulability verification for a large task system usually would take a lot of time. Such big computation cost may be acceptable for offline design, but cannot be tolerated with online systems, and so lose even more chances to adapt into different types of environments to improve overall performance. Being able to remove schedulability analysis from consideration would be very useful to improve computation efficiency.

IV. Methodology

The methodology overview is shown in Fig. 1. Each main component will be introduced in the following sub-sections, respectively.

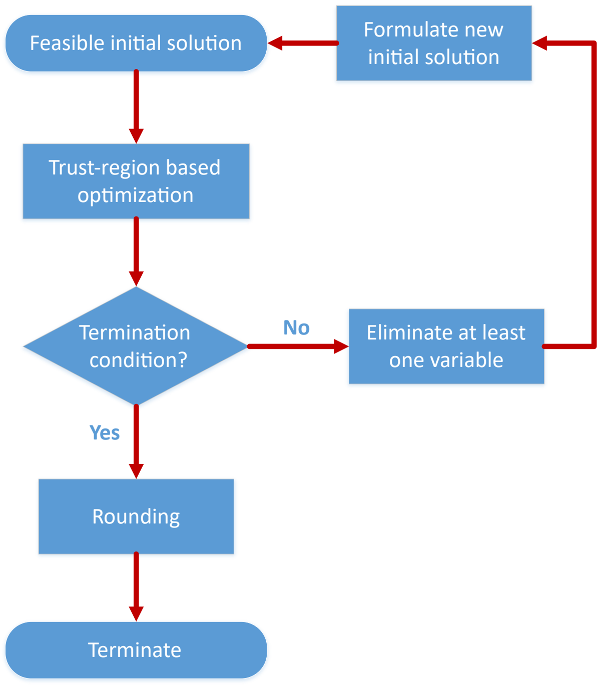
Feasible initial solution
Formulate new initial solution
Trust-region based optimization
Termination condition?
No
Eliminate at least one variable
Yes
Rounding
Terminate

Figure 1: Main optimization framework.
Begin with a feasible solution, we formulate an unconstrained least-square problem and use trust-region optimizer to solve it. After it, we check whether there are variables to eliminate. If so, we formulate a new initial solution to continue the loop. Otherwise, the algorithm returns with a local optimal result.

A. Brief review on nonlinear programming

Before presenting our methods based on nonlinear programming (NLP) to solve the problem above, we give a brief review on some classical NLP methods in case readers are not familiar with it, following the exposition in Dellaert *et al*. [9] and Boyd *et al*. [28]. Those who are already familiar with these methods can jump to the next section.

1) Nonlinear least-square problems: Let's assume we have an unconstrained least square optimization problem as follows:

$$\min_x \frac{1}{2}\|f(x) - b\|_2^2 \tag{8}$$

where $f : \mathbf{R}^n \to \mathbf{R}^m$ is continuously differentiable, but not necessarily linear nor convex.

Directly solving Problem (8) is usually very difficult, and many methods are proposed to solve this problem locally via iterations. Inside each iteration, assuming the variable is updated to be $\tilde{x}_i$, the original optimization problem is approximated as follows:

$$\min_\Delta \frac{1}{2}\|f(\tilde{x}_i) + J\Delta - b\|_2^2 \tag{9}$$

where Δ gives the updating step inside each iteration

$$\tilde{x}_{i+1} = \tilde{x}_i + \Delta \tag{10}$$

and the Jacobian matrix J is defined as

$$J_{ij} = \frac{\partial f_i(x)}{x_j} \tag{11}$$

During each iteration, the updating step Δ can be given by different methods, such as:

- Steepest descent(SD): $\Delta_{sd} = \alpha J^T b$, where the step size α could be chosen by many methods such as line search.
- Gauss-Newton(GN): $J^T J \Delta_{gn} = J^T b$, where Δ_{gn} is usually obtained via matrix decomposition and backward substitution.
- Levenberg-Marquardt (LM):
$$(J^T J + \lambda \ \mathrm{diag}(J^T J))\Delta_{lm} = J^T b \tag{12}$$
By controlling the value of scalar λ, LM can behave similarly as steepest descent or Gauss-Newton, and so take both methods' advantages.
- Dogleg(DL): Appropriate combination [29] of Steepest descent Δ_{sd} and Gauss-Newton Δ_{gn}, mostly depending on trust-region's radius and the update's gain ratio.

The last two methods can be categorized as trust-region methods. The basic idea is iteratively formulating an approximation to the original objective function, and then performing optimization within a "trust" region in each iteration. The radius of trust region is updated in each iteration, depending on how well the results that the current approximation leads to. If the objective function becomes better, the current local approximation is more likely to be consistent with the original function, and so the radius increases; on the other hand, if the objective function does not change much or even worse, that means current approximation is not good and the radius decreases. The algorithm stops when variable changes are smaller than a threshold.

2) Interior-point methods: Interior-point methods are originally introduced to handle inequality constraints in convex optimization problems [28]. The basic idea is about replacing inequality constraints with barrier functions in the objective function to enforce the optimization variables to stay within the feasibility region.

$$\text{Barrier}(u) = \begin{cases} 0, & u \le 0 \\ +\infty, & \text{otherwise} \end{cases} \tag{13}$$

In reality, it can be approximated via

$$\text{Barrier}(u) = \begin{cases} -\frac{1}{w}\log(-u), & u \le 0 \\ +\infty, & \text{otherwise} \end{cases} \tag{14}$$

where $w > 0$ is a user-defined parameter that can be adjusted to be larger iteratively so that the approximated Barrier function will be more closer to the ideal Barrier function.

For example, let's assume we have the same optimization problem as 8 but with extra inequality constraints:

$$g(x) \le 0 \tag{15}$$

We can solve the following problem iteratively to approximate the original problem, with a strictly feasible initial solution:

$$\min_x w\frac{1}{2}\|f(x) - b\|_2^2 - \frac{1}{w}\log(-g(x)) \tag{16}$$

B. Basic solution procedure

In this paper, we propose to use trust-region methods to solve the optimization problem posed by formulas 6 and 7 iteratively. To transform it into an unconstrained nonlinear optimization problem, which can be solved by the methods in the review section above, we can introduce the barrier function for each task's schedulability constraint, and then approximate the original problem as follows:

$$\min \sum_i \frac{1}{2}(f(\mathbf{c})_i - \frac{1}{w}g(\mathbf{c})_i)^2 \tag{17}$$

where $f : \mathbf{R}^N \to \mathbf{R}^N$, $g : \mathbf{R}^N \to \mathbf{R}^M$ is defined as follows:

$$\forall i \in [0, N-1], f(\mathbf{c})_i = \frac{c_i^{(0)}}{T_i} c_i^{-2} \tag{18}$$

$$\forall i \in [0, N-1], g(\mathbf{c})_i = \log(d_i - r_i) \tag{19}$$

where M is the number of inequality constraints, $w > 0$ is a weight parameter that controls relative weight of the barrier function (equivalent as t in Equation 14). We also leave a few constant parameters such as H from Equation 18. Intuition suggests that, the larger w is, the better the approximation is. This is usually true, as shown in the following sections.

To solve this problem, we can directly apply the optimization methods mentioned in the first review section. However, some changes are made to improve performance and algorithm stability based on Problem 17's nature. The overall "trust-region based optimization" framework is illustrated in Fig. 2.

Trust-region method is very suitable for optimization problems subject to real-time schedulability constraints because of the following reasons:

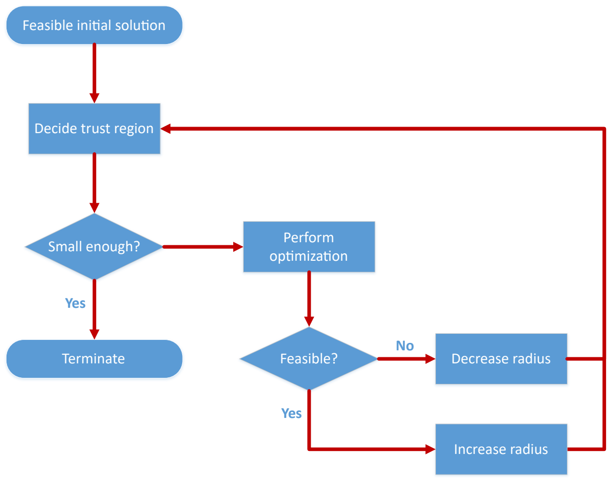

Figure 2: Trust-region based optimization. To handle discrete schedulability constraints efficiently, we only check schedulability after optimization for one step, and then modify trust-region radius accordingly.

- At first, the method always begins with a feasible point, and only accepts an update if it improves the objective function, and so the method never returns unschedulable results if the weight parameter w is chosen properly.
- When the variables are within the schedulability region, the algorithm guides the variables to move based on local approximation in each iteration. The number of iterations are usually very small compared with exponential complexity in branch-and-bound methods.
- When the variables reach out of the schedulability boundary during tentative update, the trust-region's radius shrinks until the objective function improves. Such mechanism makes the algorithm be able to handle non-linear constraints of various of forms. Furthermore, it is also a very reasonable choice for sustainable systems. If accepting one step makes the system unschedulable, then accepting a longer step will always make the system unschedulable, and so only decreasing the step could potentially give a schedulable and better result. However, as for unsustainable systems, this method will sacrifice some potential improvements.

One of the pre-requirements of applying this method is a strictly feasible initial solution. Usually, it can be estimated easily in our case: the fastest possible speed for all the tasks can serve as a good initial solution. If this initial solution is even not schedulable, then we can usually safely claim that the given system is not schedulable.

C. Jacobian matrix estimation

Since we will use gradient-based trust-region method to perform optimization, Jacobian matrix must be estimated properly. Following the chain-rule, we notice that the main difficulty lies in estimating the $\frac{\partial r_i}{\partial \mathbf{x}}$ because response time r_i is not always continuous with respect to computation time c_j.

1) Finding Jacobian matrix for discrete functions: Since discrete function is not differentiable, mathematically, the Jacobian matrix for the schedulability constraint is not well defined. As such, we first give the definition of partial derivative at a discrete point **x** within the scope of this paper:

$$J(\mathbf{x}) = (\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_N}) \tag{20}$$

where

$$\frac{\partial f}{\partial x_i} = \frac{f(x_1, ..., x_i + h, ..., x_N) - f(x_1, ..., x_i, ..., x_N)}{h} \tag{21}$$

and $h > \delta$, $\delta > 0$ is a small positive number, 0.001 in our experiments. The main difference between the given definition 21 and the classical Jacobian definition 11 is that 21 is a numerical method and only considers forward limits. Although Equation 21 is not a strict definition with elegant mathematical justification, it is able to help guide the optimization process in this paper because of the following justification:

- Necessity. The objective function monotonically decreases with respect to the optimization variables, and so variables (i.e., computation time of each task) are mostly increasing during the optimization process. (If the objective function is not monotonic, simply changing 21 from forward limits to average of forward and backward limits.) As such, it is necessary to know forward gradient information to guide the optimization process.
- Almost sufficiency. If the weight parameter is configured correctly, the absolute gradient of the objective function should always be larger than which of the barrier function because we never want the optimization variables decrease except the initial is infeasible (we will discuss this situation later). As such, all the entries of the Jacobian matrix would be negative during the optimization process and the main diagonal elements dominate. That means, usually, though probably not always, the variables will not decrease, and so there is no need to know the backward gradient.

Based on the definition in Equation 20, a slow, but universal numerical method can be directly used to evaluate the Jacobian matrix. The main computation cost happens at re-evaluating the response time, which can be speed up by beginning at a warm start.

It should be noted that even wrong estimation of the Jacobian matrix can never lead into wrong or unschedulable results because the optimizer only accepts a new step if it improves the objective function.

D. Convergence with respect to weight parameter

A natural question raised by the objective function 17 is how to set the weight parameter w appropriately. In the classical interior-point method for convex optimization problems, w can be estimated based on how accurate the results is expected because of duality gap, which bounds the difference from the given solution to optimal results. We cannot apply the similar trick in our case because the schedulability region is usually non-convex. Fortunately, the following theorem help solve this problem.

Let's denote the optimal solution of the problem specified by the minimization problem 17 as $x^*(w)$.

Theorem 1. *Given $w_1 > w_2 > 0$, we have*

$$f(x^*(w_1)) \leq f(x^*(w_2)) \tag{22}$$

Proof. Since $x^*(w)$ is already minimized, and also notice that $f(x^*(w)) - \frac{1}{w} g(x^*(w)) > 0$, we must have:

$$f(x^*(w_1)) - \frac{1}{w_1} g(x^*(w_1)) \leq f(x^*(w_2)) - \frac{1}{w_1} g(x^*(w_2)) \tag{23}$$

$$f(x^*(w_2)) - \frac{1}{w_2} g(x^*(w_2)) \leq f(x^*(w_1)) - \frac{1}{w_2} g(x^*(w_1)) \tag{24}$$

After simple manipulation, we get

$$g(x^*(w_2)) - g(x^*(w_1)) \leq w_1(f(x^*(w_2)) - f(x^*(w_1))) \tag{25}$$

$$w_2(f(x^*(w_2)) - f(x^*(w_1))) \leq g(x^*(w_2)) - g(x^*(w_1)) \tag{26}$$

And so

$$w_2(f(x^*(w_2)) - f(x^*(w_1))) \leq w_1(f(x^*(w_2)) - f(x^*(w_1))) \tag{27}$$

Considering $w_1 > w_2 > 0$, we must have

$$f(x^*(w_2)) - f(x^*(w_1)) \geq 0 \tag{28}$$

■

In English, the theorem states that the *optimal* solution of problem 17 never becomes worse as the weight parameter w increases. If the approximated problem 17 is solved by the some methods with global optimal result, and we increase the weight parameter w to infinity and it is solvable, we would get the best possible approximation result. In our case, even if we cannot provide global optimal results when solving 17, it is still more likely that the results will improve based on the logic of theorem 1.

To that end, we notice the Jacobian is estimated via

$$J_{ij} = \frac{\partial f(\mathbf{c}_i)}{\partial c_j} - \frac{1}{w} \frac{1}{d_i - r_i} \frac{\partial r_i}{c_j} \tag{29}$$

With the numerical definition for the Jacobian matrix 21 , $\frac{\partial g(\mathbf{c}_i)}{\partial c_j}$ is now guaranteed to be a limited number if the computations time variables are strictly within the schedulability region. As such, when $w \to +\infty$, we have

$$J_{ij} \approx \frac{\partial f(\mathbf{c}_i)}{\partial c_j} \tag{30}$$

Furthermore, we also can derive the following theorem:

Theorem 2. *As the weight parameter $w \to +\infty$, given the same initial solution, the optimization results for objective function 17 obtained from numerical Jacobian will converge to the results obtained from Equation 30.*

Proof. When $w \to +\infty$, J_{ij} in Equation 29 becomes arbitrarily close to Equation 30. Since each updating step during optimization solely depends on the Jacobian matrix, the final results after optimization must also converge to the result given by Equation 30.

If global optimal solution can be obtained in each step, Theorem 1 will serve as the proof for this theorem. ■

This result is extremely useful in the following two aspects:

1) Numerical robustness: One common problem that gradient-based methods face is the Jacobian matrix could be a singular, or low-rank matrix. That means, the number of constraints is less than the number of variables after linearization, and so no single solution is available. Even though the system is properly formulated initially, during iterations, some main-diagonal entries may occasionally become extremely small, or some off-diagonal entries may become much larger than the main-diagonal elements. All these situations will lead the Jacobian matrix by 21 degrade into a singular or low-rank matrix.

In our case, this usually happens when the variables are near to the schedulability boundary. When task τ_i's response time r_i is close to its deadline d_i, numerical evaluation of Jacobian $\frac{\partial r_i}{\partial c_i}$ could become a very big number. However, when w is assumed to become infinitely large, such numerical issue can be avoided because the Jacobian matrix from response time analysis part would be 0.

2) Run-time efficiency: The benefits of run-time efficiency is obvious because the Jacobian matrix in 30 is usually a diagonal matrix. That means a lot of computation savings in both Jacobian matrix's evaluation and computation parts. Especially, systems with complicated response time analysis usually do not have analytic Jacobian estimation, and numerical evaluation is usually very expensive. Removing it could achieve a big saving in computation costs. As will be shown in the experiments section, such approximation just has the same, if not better, performance as using the original numerical Jacobian matrix.

In terms of updating step, the associated saving is also big. In a large system, the computation cost of the optimization part (i.e., without considering response time analysis) mainly depends on evaluating the Jacobian matrix and performing updating. Given a fully dense $m \times n$ Jacobian matrix, the following optimization steps (such as Gauss-Newton method or its variants) requires $O(mn^2)$ or $O(n^3)$ to perform decomposition and backward substitution by methods such as QR or Cholesky decomposition. However, a diagonal Jacobian matrix as introduced before only requires linear computation cost $O(n)$, which is faster by two orders.

E. Variables elimination

In this paper, variable elimination means keeping some variables' value fixed and do not adjust or optimize them anymore in the future iterations. As mentioned above, the algorithm terminates when the variables are close to the schedulability boundary because there is not much space to improve while guaranteeing schedulability. However, there are probably still potential space to optimize for some tasks.

Example 1. Let's consider a simple example to illustrate the motivation for elimination algorithm. There are two periodic, preemptive tasks to be executed on a uni-processor system, as shown in Table I. These two tasks are scheduled by RM, the variables are the computation time, and are limited to be integers. After performing optimization for a few iterations without elimination, the variables could become very close to the schedulability boundary, as in Fig. 3. At this point, the optimization algorithm will terminate because the system will become unschedulable if keep optimizing along the gradient. However, if we remove c_1 from optimization, the gradient direction will be indicated by the blue arrow, which points toward a better local optimal solution.

Table I: Example task set for elimination

	Computation time	Period	Deadline	Offset
Task 1	4	10	10	0
Task 2	1	40	40	0

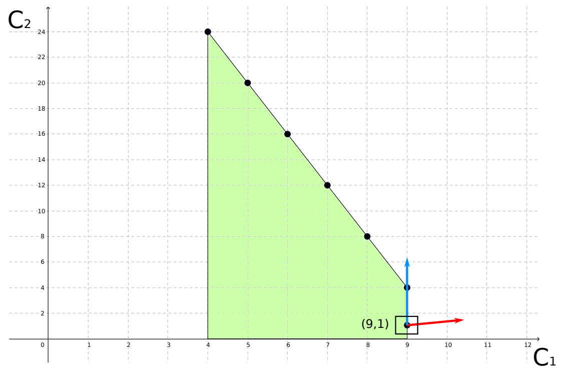

Figure 3: Visualization of elimination. The green area indicates schedulability region. At the point (9,1), the gradient before elimination is indicated by the red arrow, which leads to an unschedulable result; the gradient after elimination is indicated by the blue arrow which drives it to a better position.

Variable elimination is triggered by *either* one of the following two conditions:

$$\forall k, r_k(c_i + \delta) > d_k \tag{31}$$

$$d_i - r_i <= \mathit{Tol} \tag{32}$$

where δ represents float-type accuracy we want to achieve, *Tol* decides how close the response time of one task to its deadline could be. This condition is checked after the optimization returns, following the inverse order of priority. If τ_i is to be eliminated, then all the computation time variables of both τ_i and its high priority tasks $hep(i)$ will be eliminated. The optimization procedure continues with the rest variables (i.e., $c_l, l \in lp(i)$. The current optimization variables will be used as initial solution for the next optimization procedure.

The *Tol* variable is adjusted during optimization, if necessary, to make sure each time the optimization terminates, at least one variables will be eliminated. Otherwise, there may be deadlock loops during iterations. This method gives the termination criteria for the overall algorithm:

Theorem 3. *The overall algorithm terminates when Equation 31 or 32 is triggered for the lowest priority task.*

Proof. Each time elimination is triggered, all the higher priority tasks' computation time are eliminated. When the lowest priority task is eliminated, there is no computation tasks' computation time to adjust, and so the algorithm terminates. ■

Theorem 4. *The algorithm will always terminate.*

Proof. Since the number of tasks N are limited, and we will adjust *Tol* to make sure at least one task is eliminated, the overall algorithm will terminate within at most N loops. ■

F. Rounding float variables to integer

In reality, it is also common that there are some special types of constraints that cannot be handled by the barrier function. Continuous constraints such as range constraints can usually be incorporated by barrier function. Discrete type constraints, such as integer, harmonic requirements, or a discrete set, is more difficult. In that case, we perform continuous optimization at first and then try to round the float point type result to satisfy those discrete requirements.

Algorithm 1: Rounding continuous variables to discrete variables

```
   Input: continuous variables after optimization c
   Output: discrete variables c
1  c = floor(c)
2  taskSet = {$\tau_1, ..., \tau_n$}
3  while taskSet is not empty do
4      index = $Selection$(taskSet)
5      $c_{\text{index}} = c_{\text{index}} + 1$
6      if CheckSchedulability(c) == False then
7          $\mathbf{c}_{\text{index}} = \mathbf{c}_{\text{index}} - 1$
8          for $\tau_j \in hep(index)$ do
9              taskSet.remove($\tau_j$)
10         end
11     else
12         continue
13     end
14 end
15 return c
```

The most direct and simplest way for rounding is simply cutting the float part for all the computation variables. This method is guaranteed to return a schedulable solution for all the sustainable systems. However, such brute method may easily lead into a sub-optimal result, especially if there is a big gap among the available choices. For example, some devices only provide limited choices of discrete run-time CPU/GPU frequency. In such case, simply rounding all the computation time variables up could give a result that is far from the continuous optimization.

Aiming at the rounding issue mentioned above, and also to generalize the potential applications of the algorithm from continuous optimization into discrete optimization, we propose a simple heuristic method based on the similar idea as the continuous optimization part.

The pseudo-code is given in Algorithm 1. We begin with the most pessimistic rounding result (line 1), which satisfies the discrete requirement. The algorithm first selects one task to rounding down based on a given criterion (line 6), and then increases (rounding) its computation time towards an adjacent discrete variable. If the task set becomes unschedulable after one rounding operation, then it means this task and all its higher priority tasks cannot be optimized, and so are removed from the considered task set pool; Otherwise, the iteration continues. After the algorithm ends, it is should find an available variable that are both feasible and schedulable.

There are many criterion in deciding the *Selection* function (Line 4 in Algorithm 1). For example, it can first select the task which gives the most benefits in energy saving. Such task can be selected based on the Jacobian matrix easily. Another criteria is considering the potential cost. Rounding lower priority tasks usually has a lower cost in terms of available slack because they do not effect high priority tasks, and so should be preferred. Although we use the first method in our experiments, we suggest interested readers to consider trying different methods if the rounding could have a big influence on the overall results.

Finally, the rounding up method in line 5 should also adapt based on the problem environment. Obviously, it does not have to increase only one time unit each time. Binary search can also be applied to speed up if there is a big interval between adjacent discrete variables. Still, this part should be adjusted based on the real situations.

G. Runtime efficiency and improvements

We also want to mention that response time analysis such as Equation 31 can be performed very quickly during iterations. Usually, response time analysis with similar formulation as in Equation 1 is solved using fixed-point iterations to find the minimum r_i that satisfies the equation. This process has the following computation properties [27]:

- The estimated response time always remains the same if only the beginning time is smaller than the response time.
- A larger beginning point can never take more iterations than a smaller beginning point.

These two properties are very useful in checking schedulability during iterations. Under most situations, it only takes one iteration to find the new r_i. There are also some further tricks to speed up this process proposed by Davis *et al.* [27], though we did not consider for simplicity.

V. Application

The potential application of the proposed algorithm is far beyond what demonstrates above. In this section, we introduce some possible generalization application situations.

A. Constrains in variables

Many different types of constraints can be added by transforming them into a barrier function in the objective function. For example, if the processor can only adjust its run-time speed within a specific range, then a box constraint can be added to the objective function in such a form:

$$g(\mathbf{c})_i = \frac{-1}{w}\log(c_i - c_i^{min}) + \frac{-1}{w}\log(c_i^{max} - c_i) \tag{33}$$

If the variables are limited to be integers within a range or a discrete set of values, the rounding mechanism mentioned in the previous section can be used.

B. Different forms of objective function

Apart from the simple form of energy function used in the previous sections, more realistic, but complicated forms of energy function can be used. Static energy consumption, or different order of frequency can be easily added. If the system is very complicated, different regression methods can be used to approximate the real power consumption relationship [30], [31]. In this case, new variables can also be included, such as memory frequency. Since the results are usually continuous functions, we expect the overall optimization based on NLP methods is still able to return good results.

C. Different forms of schedulability analysis

Although we use a simple form of schedulability analysis to demonstrate the algorithm, more complicated forms of schedulability analysis can be used easily. We performed experiments based on some more complicated schedulability analysis in the experiments section for reference.

However, different forms of schedulability analysis may have a big influence on the overall run-time efficiency. During optimization, the algorithm needs to evaluate schedulability many times incrementally. As such, the overall run-time speed will improve a lot if early iterations can be used to speed up later iterations. For example, most schedulability analysis models have sustainability with respect to computation time variables. In that case, response time obtained from early iterations can serve as a warm start for later iterations, which would save a lot of computation time cost. If results obtained from different iterations are totally independent, the algorithm is expected to be less efficient, but is still able to perform well.

VI. Experiments

We performed extensive experiments based on different response time analysis models. The performance is also compared with several baseline algorithms.

A. Example system model

The first experiment shows algorithm performance for the example system model that we use before, whose response time analysis is given by Equation 1. Our algorithm is mainly compared with MUA-incremental Zhao *et al.* [8], which represents the best related work for this problem to the best of the authors' knowledge. To be comparable with Zhao *et al.* [8], we also add one more constraint that the maximum computation time of each task is no larger than two times initial computation time. The task sets and schedulability analysis that we use are exactly the same in Zhao *et al.* [8].

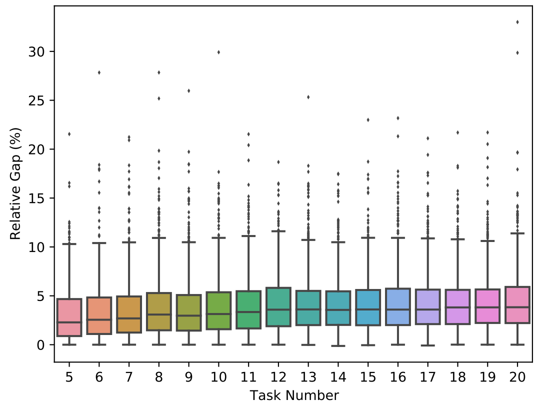

Figure 4: Comparison of Energy Optimization performance with MUA-incremental [8]. In average, the performance of our algorithm is only 2 to 3% worse.

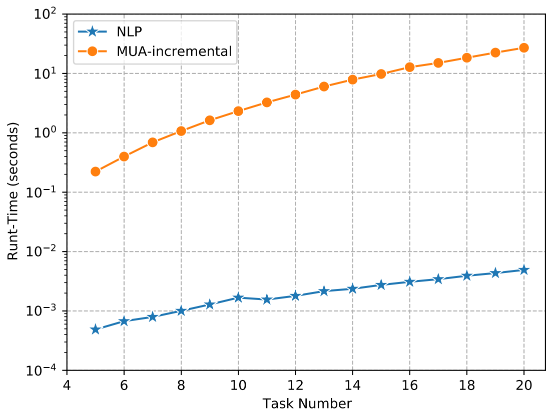

Figure 5: Comparison of running speed. NLP runs 10^3 to 10^4 times faster than MUA in experiments, and will show more advantages with even larger task sets.

The comparison of energy optimization is shown in Fig. 4, where the y-axis shows the relative gap between the proposed NLP algorithm and MUA:

$$\frac{E_{NLP} - E_{MUA}}{E_{MUA}} \times 100 \tag{34}$$

where E_{NLP}, E_{MUA} are energy consumption after optimization. The figure is plotted via box plots. As a quick remainder for box plots, $25\%-75\%$ data is plotted within the colored box region, the maximum and minimum values excluding outliers are indicated by the horizontal lines, while a very small portion of data are determined as outliers are scattered above the box. In average, the performance of the proposed algorithm is slightly worse than MUA by 3-4%, while under most cases, the difference is within 5%.

Apart from comparing with MUA, we also compare the performance of the proposed algorithm with brute-force algorithm for small task sets (task number is smaller than 5), which indicate only $1-3\%$ worse in average. As such, we believe that the performance of the proposed algorithm is very good under most situations.

The run-time speed is shown in Fig. 5. It can be easily seen that the proposed algorithm runs $1000-10000$ times faster than MUA, depending on how large the task set is. Different from MUA which has a worst-case exponential complexity, the proposed algorithm's main computation cost is dominated by response time analysis, which is well-known to have only a pseudo-polynomial complexity. Such great advantage in speed not only makes large systems easier for design and analysis, but can also be easily used for online optimization, and bring more tremendous benefits.

B. Convergence with respect to weight parameter

In this section, we did several experiments to verify the convergence with respect to barrier function's weight parameter w, given by objective function 17. Theorem 2 states that the algorithm performs better with larger w if global optimal solution can be obtained during solution process. However, in reality, it is extremely difficult to obtain global optimal solution. Usually, only local optimal solutions are available. In this section, we use experiment results based on large-scale random task sets to demonstrate that Theorem 2 still basically holds for our algorithm, even though the proposed algorithm can only obtain local optimal solutions.

The system model is the same as the one above. Without loss of generality, we only showed the result for task sets with 20 tasks because all the other task sets are very similar but simpler. The result is shown in Fig. 6, where the x-axis shows weight parameter in log scale, while the y-axis shows average energy saving ratio:

$$\text{Energy saving ratio} = \frac{E_{NLP}}{E_{init}} \tag{35}$$

When w is very small, the barrier function would have a much bigger influence than the energy function during optimization. As such, in that case, computation time variables tend to decrease rather than increase. Such situations should always be avoided in reality, and so we did not conduct experiments with very small w.

C. GPU scheduling model

We also tested the proposed algorithm under a more complicated schedulability analysis model. The task set has N periodic preemptive tasks τ_i with worst-case execution time c_i, non-ignorable preemption overhead o_i, period T_i, and deadline d_i. The scheduling action to take when one task τ_j becomes ready while another task τ_i is under execution is encoded by two binary matrices $A,\ P$. If $A(i,j) == 1$, then task τ_i always aborts task τ_j; if $A(i,j) == 0, P(i,j) == 1$, then task τ_i always preempts task τ_j; otherwise, task τ_i is always blocked by task τ_j. These two matrices are decided before analyzing schedulability. For each task τ_i, its worst-case response time r_i within its busy period $t_m^{(i)}$ is estimated as follows:

$$r_i = \max_{1 \le q \le Q} r_i^q - (q-1)T_i \tag{36}$$

where

$$r_i^q = qc_i + \text{BlockTime}_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i^q}{T_j} \right\rceil (c_j + \text{Interference}_{ji}) \tag{37}$$

$$Q = \left\lceil \frac{t_m^{(i)}}{T_i} \right\rceil \tag{38}$$

$$\text{BlockTime}_i = \max_{l \in lp(i)} \begin{cases} c_l - 1, & A_{il} == 0 \text{ and } P_{il} == 0 \\ p_l, & A_{il} == 0 \text{ and } P_{il} == 1 \\ 0, & A_{il} == 1 \end{cases} \tag{39}$$

$$\text{InterferenceH}_{ji} = \begin{cases} \max_k c_k - 1, & \exists k : i \le k < j \\ & \text{and} A_{jk} == 1 \\ o_i, & \text{otherwise and} \\ & \ P_{ji} == 1 \\ 0, & \text{otherwise} \end{cases} \tag{40}$$

From Fig. 7, 8, it can be seen that the proposed algorithm shows great advantage in terms of both energy savings and run-time speed.

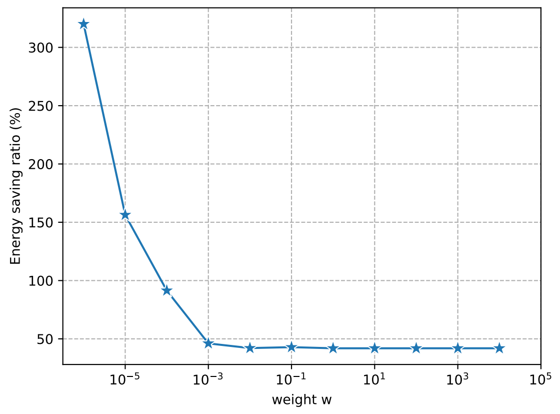

Figure 6: Convergence with respect to weight parameter w. Overall, the algorithm performs better as w increases, and does not change when w is large enough.

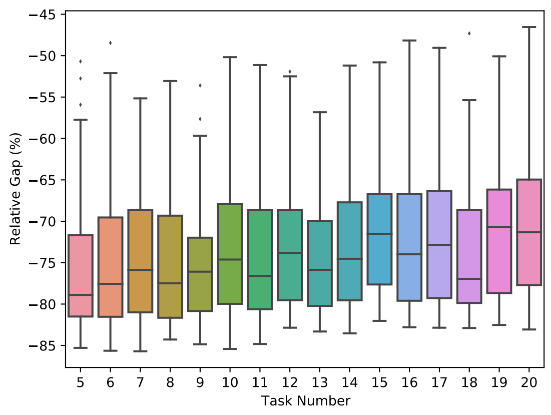

Figure 7: Comparison of Energy Optimization performance with simulated annealing, GPU scheduling model. Experiments indicate around 80% improvement compared with SA. The y-axis is measured using formula 34.

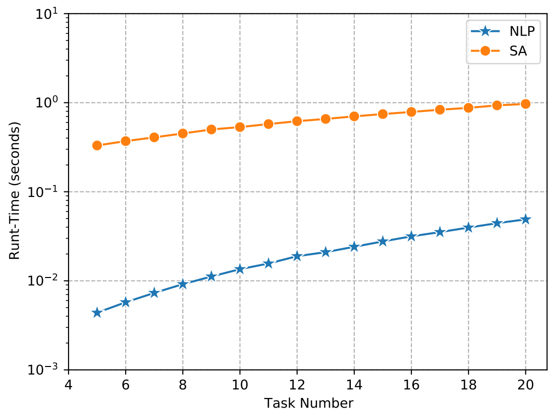

Figure 8: Comparison of Run-time speed performance with simulated annealing, GPU scheduling model. The proposed algorithm runs around 10 times faster than SA.

VII. Conclusion

In this paper, we propose a general optimization method for DVFS systems. The energy optimization problem is approximated as a least-square problem, and solved by gradient-based trust-region methods with problem-specific innovations such as variable elimination and rounding. Compared with state-of-art methods, the proposed method runs several thousands of times faster while maintaining very similar performance. The advantage in computation cost opens up new chance for online optimization, which could provide even more energy savings. Extensive experiments have been performed to test algorithm performance and related theorem.

References

[1] R. Depaola, C. Chimento, M. L. Anderson, K. Brink, and A. Willis, "Uav navigation with computer vision–flight testing a novel visual odometry technique," in *2018 AIAA Guidance, Navigation, and Control Conference*, p. 2102, 2018.

[2] K. Bazaka and M. V. Jacob, "Implantable devices: issues and challenges," *Electronics*, vol. 2, no. 1, pp. 1–34, 2013.

[3] M. Silva, D. Cerdeira, S. Pinto, and T. Gomes, "Operating systems for internet of things low-end devices: Analysis and benchmarking," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10375–10383, 2019.

[4] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo, "Energy-aware scheduling for real-time systems: A survey," *ACM Trans. Embed. Comput. Syst.*, vol. 15, pp. 7:1–7:34, 2016.

[5] P. Ekberg and W. Yi, "Fixed-priority schedulability of sporadic tasks on uniprocessors is np-hard," in *2017 IEEE Real-Time Systems Symposium, RTSS 2017, Paris, France, December 5-8, 2017*, pp. 139–146, IEEE Computer Society, 2017.

[6] H. Zeng and M. Di Natale, "An efficient formulation of the real-time feasibility region for design optimization," *IEEE Transactions on Computers*, vol. 62, no. 4, pp. 644–661, 2013.

[7] E. Bini, G. Buttazzo, and G. Lipari, "Minimizing cpu energy in real-time systems with discrete speed management," *ACM Trans. Embed. Comput. Syst.*, vol. 8, pp. 31:1–31:23, 2009.

[8] Y. Zhao, R. Zhou, and H. Zeng, "An optimization framework for real-time systems with sustainable schedulability analysis," *2020 IEEE Real-Time Systems Symposium (RTSS)*, pp. 333–344, 2020.

[9] F. Dellaert and M. Kaess, "Factor graphs for robot perception," *Found. Trends Robotics*, vol. 6, pp. 1–139, 2017.

[10] F. F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pp. 374–382, 1995.

[11] P. Pillai and K. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *Proceedings of the eighteenth ACM symposium on Operating systems principles*, 2001.

[12] A. Qadi, S. Goddard, and S. Farritor, "A dynamic voltage scaling algorithm for sporadic tasks," *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*, pp. 52–62, 2003.

[13] H. Aydin, V. Devadas, and D. Zhu, "System-level energy management for periodic real-time tasks," *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pp. 313–322, 2006.

[14] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 53, pp. 584–600, 2004.

[15] C.-H. Lee and K. Shin, "On-line dynamic voltage scaling for hard real-time systems using the edf algorithm," *25th IEEE International Real-Time Systems Symposium*, pp. 319–335, 2004.

[16] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo, "Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems," *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pp. 408–417, 2006.

[17] G. Zeng, T. Yokoyama, H. Tomiyama, and H. Takada, "Practical energy-aware scheduling for real-time multiprocessor systems," *2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 383–392, 2009.

[18] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," *Proceedings International Parallel and Distributed Processing Symposium*, pp. 9 pp.–, 2003.

[19] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez, "Determining optimal processor speeds for periodic real-time tasks with different power characteristics," *Proceedings 13th Euromicro Conference on Real-Time Systems*, pp. 225–232, 2001.

[20] M. Bambagini, F. Prosperi, M. Marinoni, and G. Buttazzo, "Energy management for tiny real-time kernels," *2011 International Conference on Energy Aware Computing*, pp. 1–6, 2011.

[21] S. Saewong and R. Rajkumar, "Coexistence of real-time and interactive & batch tasks in dvs systems," *2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 24–33, 2008.

[22] M. Shin and M. Sunwoo, "Optimal period and priority assignment for a networked control system scheduled by a fixed priority scheduling system," *International Journal of Automotive Technology*, vol. 8, pp. 39–48, 2007.

[23] K. Tindell, A. Burns, and A. Wellings, "Allocating hard real-time tasks: An np-hard problem made easy," *Real-Time Systems*, vol. 4, pp. 145–165, 2004.

[24] J. Jonsson and K. Shin, "A parametrized branch-and-bound strategy for scheduling precedence-constrained tasks on a multiprocessor system,"

Proceedings of the 1997 International Conference on Parallel Processing (Cat. No.97TB100162), pp. 158–165, 1997.
[25] M. Natale, L. Guo, H. Zeng, and A. Sangiovanni-Vincentelli, "Synthesis of multitask implementations of simulink models with minimum delays," *IEEE Transactions on Industrial Informatics*, vol. 6, pp. 637–651, 2010.
[26] N. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," 1991.
[27] R. I. Davis, A. Zabos, and A. Burns, "Efficient exact schedulability tests for fixed priority real-time systems," *IEEE Transactions on Computers*, vol. 57, pp. 1261–1276, 2008.
[28] S. P. Boyd and L. Vandenberghe, "Convex optimization," *IEEE Transactions on Automatic Control*, vol. 51, pp. 1859–1859, 2006.
[29] M. Powell, "A new algorithm for unconstrained optimization," 1970.
[30] B. Dutta, V. Adhinarayanan, and W. chun Feng, "Gpu power prediction via ensemble machine learning for dvfs space exploration," *Proceedings of the 15th ACM International Conference on Computing Frontiers*, 2018.
[31] R. Barik, N. Farooqui, B. Lewis, C. Hu, and T. Shpeisman, "A black-box approach to energy-aware scheduling on integrated cpu-gpu systems," *2016 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 70–81, 2016.