August 2004

A Design Approach for Real-Time Embedded Systems with Energy and Code Size Constraints

Insik Shin
University of Pennsylvania, ishin@cis.upenn.edu

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Sang Lyul Min
Seoul National University, symin@dandelion.snu.ac.kr

Postprint version. To appear in *Lecture Notes in Computer Science*, Real-Time and Embedded Computing Systems and Applications : 10th International Conference, RTCSA 2004.

A Design Approach for Real-Time Embedded Systems with Energy and Code Size Constraints

Abstract
Real-time embedded systems often have multiple resource constraints such as energy and code size constraints. Traditionally, techniques for reducing energy consumption for real-time embedded systems have been developed without considering code size constraints, whereas code size reduction techniques have been developed without considering energy constraints. There, however, is a tradeoff relationship between reducing dynamic energy consumption and reducing code size for real-time embedded systems. Therefore, reducing code size may result in increasing energy consumption. In this paper, we present a triple-tradeoff relationship among code size, execution time, and energy consumption and then address the code size minimization problem while considering simultaneously the energy constraints and the real-time requirements of embedded systems. We formulate such an optimization problem and prove this optimization problem is NP-hard. Given the difficulty of finding the optimal solution to the problem, we then propose four heuristic algorithms to find sub-optimal solutions and evaluate their performance through simulations.

Keywords
embedded systems, code size reduction, energy consumption, real-time, scheduling

Comments
Postprint version. To appear in *Lecture Notes in Computer Science*, Real-Time and Embedded Computing Systems and Applications : 10th International Conference, RTCSA 2004.

A Design Approach for Real-Time Embedded Systems with Energy and Code Size Constraints*

Insik Shin[1], Insup Lee[1], and Sang Lyul Min[2]

[1] Departement of Computer and Information Science
University of Pennsylvania, Philadelphia PA 19104, USA
{ishin,lee}@cis.upenn.edu
[2] School of Computer Science and Engineering
Seoul National University, Seoul, 151-742, Korea
symin@dandelion.snu.ac.kr

Abstract. Real-time embedded systems often have multiple resource constraints such as energy and code size constraints. Traditionally, techniques for reducing energy consumption for real-time embedded systems have been developed without considering code size constraints, whereas code size reduction techniques have been developed without considering energy constraints. There, however, is a tradeoff relationship between reducing dynamic energy consumption and reducing code size for real-time embedded systems. Therefore, reducing code size may result in increasing energy consumption. In this paper, we present a triple-tradeoff relationship among code size, execution time, and energy consumption and then address the code size minimization problem while considering simultaneously the energy constraints and the real-time requirements of embedded systems. We formulate such an optimization problem and prove this optimization problem is NP-hard. Given the difficulty of finding the optimal solution to the problem, we then propose four heuristic algorithms to find sub-optimal solutions and evaluate their performance through simulations.

1 Introduction

Energy consumption is one of the most important design constraints in designing battery-operated embedded systems such as digital cellular phones and personal digital assistants. Energy consumption is a critical design factor for such energy-constrained systems, since the battery operation time is a primary performance measure. Memory size is one of the most important design constraints in designing embedded systems targeting system-on-a-chip (SOC). Since the cost of a chip is proportional to the fourth (or higher) power of its die size [5], program

* This research was supported in part by NSF CCR-9988409, NSF CCR-0086147, NSF CCR-0209024, ARO DAAD19-01-1-0473, by the Ministry of Education of the Republic of Korea under the Brain Korea 21 Project in 2004, and the Ministry of Science and Technology of the Republic of Korea under the National Research Laboratory program.

code size is a key design factor that determines the memory size of a chip and thus affects the die size and the chip cost. In addition, embedded systems often have strict temporal requirements. Hence, many embedded systems can have energy and code size constraints as well as real-time constraints simultaneously. Traditionally, techniques [12, 6, 10, 1, 7, 8] for reducing energy consumption for real-time embedded systems have been developed without considering code size constraints, whereas code size reduction techniques [9] for real-time embedded systems have been developed without considering energy constraints. In this paper, we introduce a tradeoff relationship between reducing dynamic energy consumption and reducing code size for real-time embedded systems. That is, reducing code size may result in increasing energy consumption. Hence, we develop the first approach of code size minimization considering simultaneously the energy constraints and the real-time requirements of embedded systems.

For energy-constrained embedded systems, recent trends in embedded architecture provide support for dynamic voltage scaling (DVS) technique at the processor level. The CPU clock speed (and its corresponding supply voltage) can be dynamically adjusted on several commercial variable-voltage processors such as Intel's *Xscale*, AMD's *K6-2+* and Transmeta's *Crusoe* processors. This DVS technique produces a voltage vs. CPU clock speed tradeoff. Various DVS algorithms [12, 6, 10, 1, 7, 8] have been proposed on employing this tradeoff for hard real-time systems. Their main goal is to reduce the total system's energy consumption while satisfying the system's real-time requirements.

A popular code size reduction technique for embedded systems with code size constraints is to employ a "dual instruction set", with the processor capable of executing two different Instruction-Sets (IS). Compilers can reduce code size by encoding a subset of normal 32-bit instructions into a 16-bit format as in ARM Thumb [4] and MIPS16 [11]. These 16-bit instructions can be dynamically decompressed by hardware into 32-bit equivalent ones before execution. This approach can substantially reduce the code size; however, it increases the number of instructions to be executed, and thus, increases the execution time of the program. For typical examples, the compressed code may require around 70% of the space of the original code, while executing 40% more instructions [2]. Our earlier work [9] was to employ this code size vs. time tradeoff for real-time embedded systems to reduce the total system's code size.

In this paper, combining these two tradeoff relationships, we introduce a new triple-tradeoff relationship among code size, energy consumption, and execution time. From the triple-tradeoff relationship, we can see that techniques to reduce code size may result in increasing energy consumption in real-time systems. Using the triple-tradeoff relationship, we address a design problem for real-time embedded systems with energy and code size constraints. The design problem, called the *code size optimization problem*, is to minimize the system's total code size satisfying the real-time and energy constraints imposed on the system. To the best of the authors' knowledge, there is no previous work that addresses the code size minimization problem considering the real-time and energy constraints simultaneously. We formulate the code size optimization problem and prove the

problem is NP-hard. Given the intractability of the problem, we propose heuristic algorithms to find sub-optimal solutions and evaluate them through simulations.

The rest of this paper is organized as follows: Section 2 presents related work about energy reduction techniques and code size reduction techniques. Section 3 provides the problem description and the system model. Section 4 formulates the optimization problem and presents heuristic algorithms. Section 5 evaluates the performance of the heuristic solutions with simulation results. Finally, we conclude in Section 6 with discussion on future research.

2 Related Work

2.1 Energy Reduction Techniques

Energy consumption is one of the most important design factors for battery-oriented embedded systems. The dynamic power dissipation P_D, which is the dominant component of energy consumption in widely popular CMOS technology, is given by $P_D \propto C_L \cdot V_{DD}^2 \cdot F_{clock}$, where C_L is the load capacitance, V_{DD} is the supply voltage, and F_{clock} is the clock frequency. Because the dynamic power dissipation P_D is quadratically dependent on the supply voltage V_{DD}, lowering V_{DD} is an effective technique in reducing the energy consumption. However, lowering the supply voltage also decreases the clock speed, because the circuit delay T_D of CMOS circuits is given by $T_D \propto V_{DD}/(V_{DD} - V_T)^\alpha$, where V_T is the threshold voltage and α is a technology-dependent constant. Hence, lowering V_{DD} yields a tradeoff between reducing the energy consumption and increasing the circuit delay. The dynamic voltage scaling (DVS) technique, which involves dynamically adjusting the supply voltage and the CPU clock speed, has been widely accepted as a key technique to reduce the energy consumption for embedded systems. This technique has been increasingly employed on commercial variable-voltage processors such as Intel's *Xscale*, AMD's *K6-2+* and Transmeta's *Crusoe* processors. Targeting these processors, various DVS algorithms [12, 6, 10, 1, 7, 8] have been proposed to reduce the dynamic energy consumption while satisfying the system's real-time requirements. One of the main differences between these approaches and our approach is that they considered only energy and temporal constraints, but we consider code size, energy, and temporal constraints.

2.2 Code Size Reduction Techniques

For many embedded systems, program code size is a critical design factor. We present a brief overview of a compiler technique for code size reduction that works for a processor capable of executing dual bit-width instructions. A very good example of such a processor is ARM microprocessors with a 32-bit instruction set (IS) for normal modes and a 16-bit reduced bit-width IS for Thumb modes [4]. A reduction in code size comes from encoding a subset of the 32-bit normal mode IS into the 16-bit Thumb mode IS. At the execution time,

a decompression engine converts a Thumb-mode instruction into an equivalent normal-mode instruction during the decode stage. The Thumb IS can access only 8 general purpose registers (out of 16 general purpose registers in the normal mode) and can encode only a small immediate value. These limitations increase the number of execution cycles and, thus, increases the program execution time. For typical programs, by using this technique the code size can be reduced by around 30%, while the number of execution cycles increases by about 40% [2].

The dual bit-width ISA allows a program to contain both 32-bit normal-mode instructions and 16-bit reduced bit-width instructions where the mode change between the two can be performed by executing a single mode-change instruction. This capability allows for a tradeoff between code size and the number of execution cycles when compiling a program. For example, by progressively transforming program units such as functions or basic blocks in the normal mode into the equivalent ones in the reduced bit-width mode while adding patch-up code to maintain the correct semantics, we can obtain a table that gives possible (code size, the number of execution cycles) pairs. The order by which the transformation is performed considers both reduction in code size and increase in the the number of execution cycles, i.e., it favors program units that give large reduction in code size with only a small increase in the number of execution cycles. Our earlier work [9] proposed a design framework that deals with a design problem taking advantage of this code size vs. time tradeoff. In this paper, we introduce a new design approach that simultaneously considers code size, energy, and temporal constraints.

3 Problem Description and System Model

3.1 Problem Description

Embedded systems often have resource constraints as well as real-time requirements. In this paper, we consider two resource constraints that are energy and code size constraints. The dynamic voltage scaling (DVS) technique yields a tradeoff relationship between voltage and CPU clock speed for the system, and with this tradeoff, we can easily build an energy consumption vs. execution time tradeoff for a task. Code size reduction techniques such as one using the dual instruction sets produce a tradeoff relationship between code size and execution time for an individual task. Combining these two tradeoff relationships, we present a new triple-tradeoff relationship among code size, execution time, and energy consumption of a task. We address a design problem for real-time embedded systems with this triple-tradeoff relationship. The design problem, called *code size optimization problem*, is to determine the code size s, the number of execution cycles n, and the CPU clock speed f of each real-time task such that

- at compile time, compilers can employ s and n to generate its executable code in order to minimize the system's total code size, and
- at run time, the EDF task scheduler can use the CPU frequency f to schedule the task in order to satisfy its real-time requirement and maintain the system's energy consumption below its upper bound.

3.2 System Model

Task Parameters. We assume that a real-time embedded system τ_{sys} is composed of a set of real-time tasks, $\{\tau_1, \ldots, \tau_n\}$ and we refer $|\tau_{sys}|$ to the number of real-time tasks in the system. We consider a real-time task has the following parameters:

- *period P:* the fixed time interval between the arrival times of two consecutive requests of the task,
- *deadline D:* the time instant relative to the start-of-its-period by which the task's execution is required to be completed ($D = P$),
- *the number of execution cycles n:* the number of execution cycles required to complete its execution in the worst case,
- *CPU frequency f:* the CPU frequency (or clock speed) at which the task executes ($F_{min} \leq f \leq F_{max}$, where F_{max} and F_{min} denote the maximum and minimum CPU frequencies available in the system, respectively),
- *execution time t:* the time amount required to complete the task's execution at the CPU frequency of f in the worst case; t is determined as n/f,
- *energy consumption e:* is the amount of energy consumed during a single execution of the task; e is determined as $t \cdot v^2 \cdot f$ where v is the supply voltage depending on f, and
- *code size s:* the size of the task's executable code.

System parameters. We define the system code size (s_{sys}) as the sum of the code sizes of all the tasks in the system, i.e.,

$$s_{sys} = \sum_{\tau_i \in \tau_{sys}} s_i.$$

We also define the system energy consumption (e_{sys}) as the sum of energy consumption of all the tasks in the system during a hyper-period, i.e.,

$$e_{sys} = \sum_{\tau_i \in \tau_{sys}} \frac{P_{LCM}}{P_i} \cdot e_i,$$

where the hyper-period P_{LCM} is the least common multiplier (LCM) of P_i's of all τ_i's in the system. We refer e_{sys}^{max} to the maximum available e_{sys} and e_{sys}^{min} to the minimum available e_{sys}. We define the range of the system energy consumption e_{sys}^{range} as $(e_{sys}^{max} - e_{sys}^{min})$.

Size/Time Tradeoff. We assume that each real-time task has a tradeoff relationship between code size and execution time. In this paper, we assume that the tradeoff relationship is given as multiple pairs of possible (code size (s), the number of execution cycles (n)) values. We refer *size/time* tradeoff to this tradeoff and M_{ST} to the number of tradeoff pairs. Figure 1 shows examples of size/time tradeoff pairs of three tasks. For each task τ_i, we sort these size/time tradeoff pairs in a decreasing order of code size and refer $S_{i,j}$ and $N_{i,j}$ to the

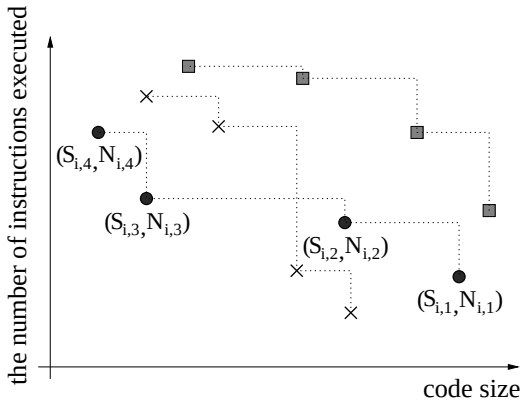

Fig. 1. Examples of task-level tradeoff between size and time

Speed Level	Speed (MHz)	Voltage (V)
1	400	1.0
2	600	1.3
3	800	1.6
4	1000	1.8

Table 1. CPU clock speed settings and voltages

code size and the number of execution cycles in the $j-th$ pair, respectively. In Figure 1, four circles represents the tradeoff pairs of task τ_i and the four circles are named as $(S_{i,1}, N_{i,1}), \ldots,$ and $(S_{i,4}, N_{i,4})$ in the decreasing order of code size, respectively. As shown in the figure, the size/time tradeoff pairs may not be clearly modeled as a continuous convex, linear, or concave function. Hence, in this paper, we consider a discrete step function $\mathtt{ST}_i$ that captures the possible (s, n) pairs of task τ_i as follows:

$$n = \mathtt{ST}_i(s), \qquad \text{where } s \in \{S_{i,1}, \ldots, S_{i,M_{ST}}\}, \tag{1}$$

Figure 1 also shows examples of discrete step functions that captures the tradeoff pairs of three tasks, respectively.

Energy/Time Tradeoff. We also assume that a real-time embedded system has a tradeoff relationship between voltage and CPU clock speed. In this paper, we assume that the tradeoff relationship is given as one in Table 1, where $F_1 = 400MHz, \ldots, F_4 = F_{max} = 1GHz$ and $V_1 = 1.0V, \ldots, V_4 = 1.8V$, respectively. We refer *voltage/speed* tradeoff to this tradeoff and M_{FV} to the number of tradeoff pairs. We consider another discrete step function FV that captures

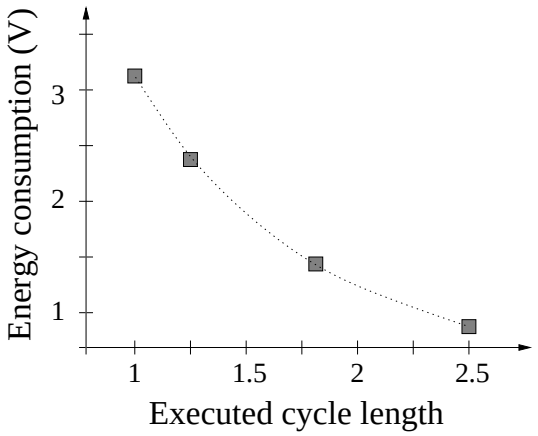

Fig. 2. System-level tradeoff between energy consumption and execution time

this tradeoff relationship as follows:

$$v = \texttt{FV}(f), \qquad \text{where } f \in \{F_1, \ldots, F_{max}\}, \tag{2}$$

Given n_i and f_i for task τ_i, we can compute the execution time and energy consumption of the task τ_i as follows:

$$t_i = n_i/f_i \quad \text{and} \quad e_i = t_i \cdot \texttt{FV}(f_i)^2 \cdot f_i = n_i \cdot \texttt{FV}(f_i)^2.$$

With the voltage/speed tradeoff given in Table 1, we build a tradeoff between energy consumption and execution time for any task and Figure 2 shows such a tradeoff for one execution cycle for all possible f_i's. We refer *energy/time* tradeoff to this tradeoff.

Size/Time/Energy Triple-Tradeoff. So far, we considered the size/time tradeoff and the energy/time tradeoff, respectively. Combining these two tradeoffs, we now consider a new tradeoff. For each task τ_i, we construct a tradeoff relationship among code size s_i, execution time t_i, and energy consumption e_i. To capture such a triple-tradeoff relationship of task τ_i, we define a *triple-tradeoff tuple* $X_i(s_i, t_i, e_i)$. We create this triple-tradeoff relationship as follows: (1) we first obtain the cartesian product of two sets, one of which is a set of the number of execution cycles $\{n_i\}$ and the other of which is a set of the CPU frequency $\{f_i\}$, and then define a triple-tradeoff tuple $X_i(s_i, t_i, e_i)$ for each element (n_i, f_i) of the cartesian product result $\{(n_i, f_i)\}$ as follows:

$$s_i = \texttt{ST}_i^{-1}(n_i), \quad t_i = \frac{n_i}{f_i}, \quad \text{and} \quad e_i = n_i \cdot \texttt{FV}(f_i)^2,$$

where $n_i = N_{i,1}, \ldots, N_{i,M_{ST}}$, and $f_i = F_1, \ldots, F_{max}$. The number of possible values of $X_i(s_i, t_i, e_i)$ is simply $M_{ST} \cdot M_{FV}$, where M_{FV} is the number of the voltage/speed tradeoff values.

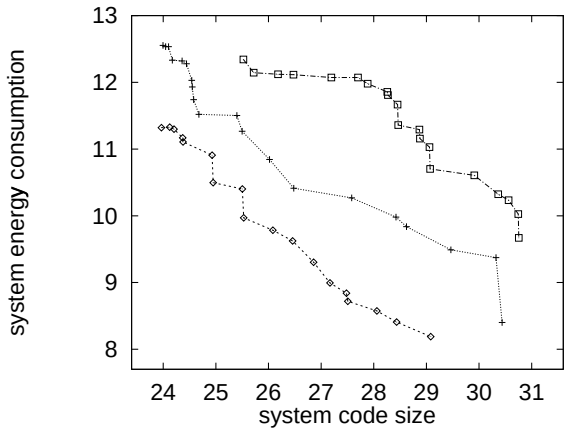

Fig. 3. Examples of system code size vs. system energy consumption tradeoff

When each task τ_i has triple-tradeoff tuples $X_i(s_i, t_i, e_i)$, we can consider a set of triple-tradeoff tuples $X = \{X_i\}$ such that X includes exactly one triple-tradeoff tuple X_i for each task τ_i. From all possible triple-trade tuple sets, we now introduce a new relationship between the system code size and the system energy consumption. For each triple-tradeoff tuple set X, we can compute its system code size s_{sys} and its system energy consumption e_{sys} and can check whether its satisfies system's real-time constraints, if given. For two triple-tradeoff tuple sets X and X' both of which satisfy some given system's real-time constraints, we define a partial order such that X is said to be smaller than X', denoted by $X < X'$, if $s_{sys} < s'_{sys}$ and $e_{sys} < e'_{sys}$. A triple-tradeoff tuple set X is said to be a *boundary* triple-tradeoff tuple set if there is no smaller triple-tradeoff tuple set X' than X, i.e., if $X' \not< X$ for all X''s that satisfy some given system's real-time constraints. In Figure 3, we plot three typical example pairs of (s_{sys}, e_{sys}) that are obtained from boundary triple-tradeoff tuple sets through simulations.[1] The figure shows that decreasing the system code size generally increases the system energy consumption.

4 Problem Formulation and Heuristic Algorithms

In this section, we formulate our code size optimization problem and prove this problem is NP-hard. Given the difficulty of finding the optimal solution to the problem, we present four heuristic algorithms that find sub-optimal solutions.

4.1 Problem Formulation and Complexity

We formulate the code size optimization problem as follows:

[1] A detail of simulation settings is described in Section 5.

– Objective:

$$\text{minimize} \sum_{\tau_i \in \tau_{sys}} s_i.$$

– Constraints:

$$\text{real-time constraint:} \quad \forall \tau_i \in \tau_{sys} : t_i \leq P_i \quad \text{and} \quad \sum_{\tau_i \in \tau_{sys}} \frac{t_i}{P_i} \leq 1,$$

$$\text{energy constraint:} \quad e_{sys} \leq E_{sys}^{upper}.$$

We present the following theorem to show that the code size optimization problem is intractable.

Theorem 1. *The problem of finding the minimum system code size satisfying the system's real-time and energy constraints is NP-hard.*

Proof. The proof is via a polynomial-time reduction from the subset sum problem that is known to be NP-complete [3]. Let a set of positive integers $A = \{a_1, \ldots, a_k\}$ and g represent an instance of the subset sum problem that is to find a subset $A' \subseteq A$ such that $\sum_{a \in A'} a = g$. Assume that for each i, $1 \leq i \leq k$, $a_i \geq 1$ and $\sum_{i=1}^{k} a_i = M$.

For reduction, we construct a set of tasks $\tau_{sys} = \{\tau_i, \ldots, \tau_k\}$ as follows: for each task τ_i, $1 \leq i \leq k$, we first construct its period such that $P_i = k \cdot B + g$ for some integer $B > 0$ and then construct its size/time tradeoff pairs in the form of (s_i, n_i) as $(a_i + B, B)$ and $(B, a_i + B)$. We also construct a voltage/speed tradeoff in the form of (v_i, f_i) as (1,1) and (2,2). Combining these two tradeoffs, we can get a triple-tradeoff tuple in the form of (s_i, t_i, e_i) as $(a_i + B, B, B)$, $(a_i + B, B/2, 2 \cdot B)$, $(B, a_i + B, a_i + B)$, and $(B, (a_i + B)/2, 2 \cdot (a_i + B))$, with $t_i = n_i / f_i$ and $e_i = n_i \cdot v_i^2$. Finally, we construct the upper bound to the system energy consumption e_{sys}^{upper} as $k \cdot B + g$.

Consider that in any solution,

$$\sum_{i=1}^{k} (s_i + n_i) = 2 \cdot k \cdot B + M. \tag{3}$$

That is, $\sum_{i=1}^{k} s_i$ is minimized when $\sum_{i=1}^{k} n_i$ is maximized. In any solution that satisfies the system's real-time and energy constraints, $\sum_{i=1}^{k} t_i \leq k \cdot B + g$ and $\sum_{i=1}^{k} e_i \leq k \cdot B + g$. Add these two inequalities, we get

$$\sum_{i=1}^{k} t_i + e_i \quad = \quad \sum_{i=1}^{k} n_i \cdot (\frac{1}{f_i} + v_i^2) \quad \leq \quad 2 \cdot k \cdot B + 2 \cdot g. \tag{4}$$

Since $\sum_{i=1}^{k} s_i$ is minimized when $\sum_{i=1}^{k} n_i$ is maximized, the problem of minimizing $\sum_{i=1}^{k} s_i$ is equivalent to the problem of maximizing $\sum_{i=1}^{k} n_i$. From Eq. (4), we can see that when $(1/f_i + v_i^2)$ is minimized for each τ_i, $\sum_{i=1}^{k} n_i$ is maximized. From the voltage/speed tradeoff, we know that when $v_i = f_i = 1$, $(1/f_i + v_i^2)$ is

minimized to 2 for each τ_i. With this, we get $\sum_{i=1}^{k} n_i \leq k \cdot B + g$ from Eq (4). When $\sum_{i=1}^{k} n_i$ is maximized to $k \cdot B + g$, $\sum_{i=1}^{k} s_i$ is minimized to $k \cdot B + M - k$ from Eq. (3). Hence, the minimum system code size is achieved if and only if there is a subset τ'_{sys} of τ_{sys} such that $\sum_{\tau_i \in \tau'_{sys}} (n_i - B) = g$; for task $\tau_i \in \tau'_{sys}$, we set n_i to $a_i + B$ and for $\tau_j \notin \tau'_{sys}$, we set n_j to B, respectively. The problem of finding such a subset τ'_{sys} is equivalent to the problem of finding the subset A' of A such that $\sum_{a \in A'} a = g$. It is obvious that the reduction can be done in polynomial time.

4.2 Heuristic Algorithms

Given the intractability of the code size optimization problem, a natural approach to resolve the problem is to develop heuristic algorithms that find suboptimal solutions. Basically, we consider heuristic algorithms that gradually reduce the system code size by increasing the number of execution cycles of a task and adjusting the clock speed for the task as long as the system's real-time and energy constraints are satisfied. Our algorithms commonly have two steps: initialization and code size reduction. In the initialization step, the algorithms find a set of triple-tradeoff tuples $X = \{(s_i, t_i, e_i)\}$ for all tasks τ_i such that the system's double constraints are satisfied with X. In the code size reduction step, the algorithms change X to X' to reduce the system code size while still satisfying the system's double constraints.

Initialization Step. The goal of this step is to initialize a set of triple-tradeoff tuples of all tasks such that the collective tuples X satisfy the system's real-time and energy constraints.

Each task τ_i has a minimum available execution time t_i^*, when its has the smallest number of execution cycles $N_{i,1}$ to execute at the fastest clock speed F_{max}, i.e., $n_i = N_{i,1}$ and $f_i = F_{max}$. We assume that the system's real-time requirements are satisfied when each task τ_i has the minimum possible execution time t_i^*, i.e., $X\{s_i, t_i^*, e_i\}$. Otherwise, there is no feasible solution to the code size optimization problem satisfying the system's real-time constraints.

For each task τ_i, this step initializes n_i and f_i as follows:

$$n_i = N_{i,1} \quad \text{and} \quad f_i = F_{max}.$$

Each task τ_i then has its triple-tradeoff tuple (s_i, t_i, e_i) initialized as follows:

$$s_i = S_{i,1}, \quad t_i = N_{i,1}, \quad \text{and} \quad e_i = N_{i,1} \cdot FV(F_{max})^2.$$

For these tuples (s_i, t_i, e_i), t_i is initialized to its minimum possible execution time t_i^*. By our assumption, the collective tuples (s_i, t_i, e_i) thus satisfy the system's real-time constraints. Unlike t_i, e_i is not initialized to its minimum possible value. Thus, the collective tuples may not satisfy the system energy constraints.

If the collective tuples satisfy the system energy constraints, the system's real-time and energy constraints are both satisfied and this initialization step finishes.

Otherwise, this step decreases the system energy consumption by changing X to X' as follows, until the system's double constraints are satisfied. It selects a task τ_i to change its tuple $X_i(s_i, t_i, e_i)$ to $X_i'(s_i', t_i', e_i')$. Our initialization step selects a task with the highest ratio $\Delta_{e_i}/\delta_{t_i}$, where

$$\Delta_{t_i} = \frac{t_i' - t_i}{P_i} \quad \text{and} \quad \Delta_{e_i} = \frac{e_i - e_i'}{P_i}.$$

Code Size Reduction Step. The initialization step makes a set of triple-tradeoff tuples of all tasks (X) satisfy the system's real-time and energy constraints. Given X from the initialization step, the goal of this step is to adjust X to reduce the system code size while the two system constraints are still satisfied. We consider four heuristic algorithms to achieve such a goal.

The four heuristic algorithms commonly to the following procedures: they select a task τ_i and adjust its task tradeoff variables from $X_i(s_i, t_i, e_i)$ to $X_i'(s_i', t_i', e_i')$ in order to decrease the system code size without violating either of the system's real-time and energy constraints. They repeat this sub-step until there is no eligible task to decrease its code size without violating any constraint.

The four heuristic algorithms differ from each other in the selection of a task to decrease a code size. We first introduce two naive heuristic algorithms, *TM-ONLY* and *EZ-ONLY*. These two algorithms consider only one aspect of the system's real-time and energy constraints, respectively.

- The TM-ONLY algorithm favors a task that can efficiently reduce a code size without increasing an execution time much. The TM-ONLY algorithm chooses a task with the highest ratio $\Delta_{s_i}/\Delta_{t_i}$, where

$$\Delta_{s_i} = s_i - s_i', \quad \text{and} \quad \Delta_{t_i} = \frac{t_i' - t_i}{P_i}.$$

- The EZ-ONLY algorithm favors a task that can efficiently reduce a code size without increasing an energy consumption much. The EZ-ONLY algorithm chooses a task with the highest ratio $\Delta_{s_i}/\Delta_{e_i}$, where

$$\Delta_{s_i} = s_i - s_i', \quad \text{and} \quad \Delta_{e_i} = \frac{e_i' - e_i}{P_i}.$$

Now, we introduce two other heuristic algorithms, *FIX-MIX* and *DYN-MIX*, that simultaneously consider the both aspects of the system's double constraints.

- The FIX-MIX algorithm favors a task that can efficiently reduce its code size without much increasing both an execution time and an energy consumption. To consider an execution time increase and an energy consumption increase together, we normalize them. Let $\Delta_{t_i}^*$ denote a normalized execution time increase of τ_i and $\Delta_{e_i}^*$ denote a normalized energy consumption increase of τ_i such that

$$\Delta_{t_i}^* = \frac{\Delta_{t_i}}{P_{LCM}} \quad \text{and} \quad \Delta_{e_i}^* = \frac{\Delta_{e_i}}{e_{sys}^{max} - e_{sys}^{min}}$$

  The FIX-MIX algorithm chooses a task with the highest ratio $\Delta_{s_i}/(\Delta_{t_i}^* + \Delta_{e_i}^*)$.

Between the system's double constraints, one constraint may be tight while the other is loose. In this situation, a reasonable approach is to favor a task that can reduce a code size by placing the least impact on the tighter constraint and by imposing a large impact on the looser constraint. Considering this, we develop the DYN-MIX algorithm as follows.

- The DYN-MIX algorithm favors a task that can efficiently reduce a code size without much increasing either an execution time or an energy consumption, whichever has a tighter system constraint. To capture the tightness of the constraints, we define *time weight* (w_t) and *energy weight* (w_e) as follows: w_t represents the normalized distance of the current CPU utilization to the upper-bound constraint to the CPU utilization, i.e., 1, and w_e represents the normalized distance of the current system energy consumption (e_{sys}) to the upper-bound constraint to the system energy consumption (E_{sys}^{upper}), i.e.,

$$w_t = \sum_{\tau_i \in \tau_{sys}} \frac{t_i}{P_i} \quad \text{and} \quad w_e = \frac{E_{sys}^{upper} - e_{sys}}{E_{sys}^{upper} - e_{sys}^{min}}.$$

  The time weight w_t and energy weight w_e dynamically change reflecting the tightness of the real-time and energy constraints. The DYN-MIX algorithm chooses a task with the highest ratio $\Delta_{s_i}/(w_t \cdot \Delta_{t_i}^* + w_e \cdot \Delta_{e_i}^*)$.

5 Simulation Results

For the purpose of evaluating the performances of heuristic algorithms, we performed simulations on finding the optimal solution and some heuristic solutions to the code size optimization problem. We first describe the simulation parameter configuration that was used during the simulations and then evaluate the performances of heuristic algorithms.

5.1 Simulation Parameters

During simulations, our performance measure is the error of a heuristic solution to the reference solution in terms of the system code size. We define the error $E(s, s^0, s^*)$ of a solution s from the initial solution s^0 to the reference solution s^* as follows:

$$\texttt{Err}(s, s^0, s^*) = \frac{s - s^*}{s^0 - s^*}. \tag{5}$$

During simulations, the initial solution s^0 is the initial (maximum) system code size. The reference solution s^* is the optimal solution obtained through an exhaustive search or the best heuristic solution among the heuristic solutions in simulations.

During simulations, we have the following simulation parameters:

- The number of tasks ($|\tau_{sys}|$) is determined in the range [5,150].

- The period (P_i) of a task τ_i is randomly generated in the range [100, 1000].
- The size/time tradeoff of each task τ_i is generated as a sorted list of (code size, the number of execution cycles) pairs. That is, the tradeoff of τ_i is given as $\{(S_{i,1}, N_{i,1}), \ldots, (S_{i,M_{ST}}, N_{i,M_{ST}})\}$, where $S_{i,k} > S_{i,k+1}$ and $N_{i,k} < N_{i,k+1}$ $(1 \leq k \leq M_{ST} - 1)$.
  - The number of tradeoff values (M_{ST}) is determined in the range [4, 50].
  - The maximum possible code size ($S_{i,1}$) is randomly generated in the range [1000, 10000]. The minimum possible code size ($S_{i,M_{ST}}$) is also randomly generated in the range $[0.7 \cdot S_{i,1}, 0.9 \cdot S_{i,1}]$ to reflect the code size reduction technique's performance for typical programs [2]. The intermediate code size values, $S_{i,2}, \ldots, S_{i,M_{ST}-1}$, are randomly generated in the range $[S_{i,1}, S_{i,M_{ST}}]$ such that they are unique and sorted.
  - The minimum number of execution cycles ($N_{i,1}$) is randomly generated in the range $[1,(3 \cdot P_i)/|\tau_{sys}|]$, where $|\tau_{sys}|$ is the number of tasks. The maximum number of execution cycles ($N_{i,M_{ST}}$) is also randomly generated in the range $[1.1 \cdot N_{i,1}, 1.4 \cdot N_{i,1}]$ following the reduction technique's typical performance [2]. The intermediate values, $N_{i,2}, \ldots, N_{i,M_{ST}-1}$, are randomly determined in the range $[N_{i,1}, N_{i,M_{ST}}]$ such that they are unique and sorted.
- For an energy/time tradeoff, the voltage/speed tradeoff shown in Table 1 is used in the simulations. To place an energy constraint on the system, the upper-bound of the system energy consumption (E_{sys}^{upper}) is randomly generated in the range $[e_{sys}^{min} + 0.15 \cdot e_{sys}^{range}, e_{sys}^{min} + 0.85 \cdot e_{sys}^{range}]$. To place a real-time constraint on the system, a task set is used for simulations if and only if the task set is schedulable when $n_i = N_{i,1}$ and $f_i = F_{max}$ for each task τ_i, but is not schedulable when $n_i = N_{i,M_{ST}}$ and $f_i = F_1$.

In addition to the four heuristic algorithms introduced in Section 4.2, we have another greedy heuristic algorithm *RANDOM* that randomly selects a task to reduce its code size among all tasks as long as the system's real-time and energy constraints are satisfied. While performing simulations, we did not consider a simulation run as valid if our four heuristic solutions and RANDOM's solution are identical in the simulation run. Excluding these simulation runs where the system's real-time and energy constraints are both so loose that any greedy algorithm can have the same solution, we performed 500 or 1000 simulation runs for each simulation setting.

5.2 Heuristic Solution Evaluation

First, we evaluate the heuristic solutions comparing them with the optimal solution. The error of heuristic solutions are calculated taking the optimal solution as their reference solution. Due to the complexity of the exhaustive search to find the optimal solution, we limited the number of tasks to 5. The number of size/time tradeoff pairs M_{ST} increases from 4 to 12 by 2. Figure 4 plots the average error of heuristic solutions for 500 simulation results as a function of M_{ST}.

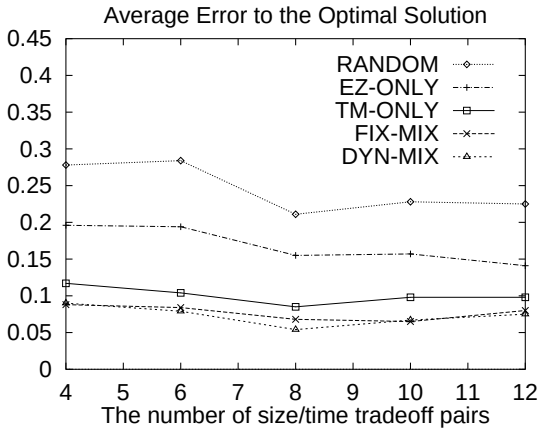

Fig. 4. Average error of heuristic solutions to the optimal solution as a function of the number of size/time tradeoff pairs (M_{ST}).

Sim. Type		DYN-MIX	FIX-MIX	TM-ONLY	EZ-ONLY	RANDOM
I	Mean	0.073	0.078	0.101	0.177	0.252
	Std. Dev.	0.120	0.122	0.136	0.186	0.209
II	Mean	0.002	0.004	0.008	0.224	0.415
	Std. Dev.	0.011	0.011	0.019	0.156	0.198
III	Mean	0.001	0.004	0.007	0.374	0.493
	Std. Dev.	0.013	0.017	0.026	0.228	0.218

Table 2. Statistical analysis of simulation results

Figure 4 shows that DYN-MIX and FIX-MIX have considerably better performance than the other heuristic solutions. This implies that we can significantly reduce the system code size while considering the both aspects of the system's real-time and energy constraints at the same time. The mean and standard deviation of the error of heuristic solutions are shown in Table 2 under simulation type I. The results in the table shows that DYN-MIX has the best performance among all the heuristic algorithms. This result implies that we can achieve a better performance in considering the tightness of the system's constraints.

Due to the complexity of the exhaustive search, we could not find the optimal solutions when performing simulations with a large number of size/time tradeoff values and/or a large number of tasks. In this case, we used the best heuristic solution as the reference solution, rather than the optimal solution, in comparing the heuristic solutions. The average errors of the heuristic solutions obtained through 1000 simulation results are plotted as a function of M_{ST} while $|\tau_{sys} = 10|$ in Figure 5, and as a function of τ_{sys} while $M_{ST} = 10$ in Figure 6, respectively. It is shown in the both figures that DYN-MIX, FIX-MIX, and TM-ONLY significantly outperforms EZ-ONLY and RANDOM and that the

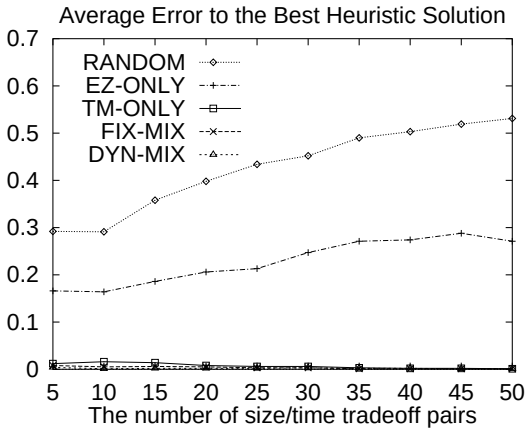

Fig. 5. Average error of heuristic solutions to the best heuristic solution as a function of the number of size/time tradeoff pairs (M_{ST}).

performance gap between these two groups increases as M_{ST} and $|\tau_{sys}|$ increase, respectively. The mean and standard deviation of heuristic solutions' errors for variable M_{ST} cases are shown in Table 2 under simulation type II, and those for variable $|\tau_{sys}|$ are shown in the same table under simulation type III. These simulation results consistently show that DYN-MIX is the most promising heuristic algorithm among the five ones.

6 Conclusion

Combining the code size vs. execution time tradeoff and the voltage vs. clock speed tradeoff, we introduced a triple-tradeoff among code size, execution time, and energy consumption. With this triple-tradeoff, we addressed the problem of minimizing code size for energy-constrained real-time embedded systems. We presented a mathematical model of the code size optimization problem and then proved the problem is NP-hard. With the intractability of the code size optimization problem, we presented heuristic algorithms that find sub-optimal solutions. The implication of our simulation results is that heuristic algorithms yield better solutions when considering the system's real-time and energy constraints simultaneously. In particular, the simulation results show that it is worthy for heuristic appraoches to consider the tightness of the both constraints.

In this paper, we assume that the code size vs. execution time tradeoff relationship and the voltage vs. clock speed tradeoff relationship are both discrete. However, these two tradeoff relationships can be approximated by continuous functions. Our future work is to develop a design framework that deals with design optimization problems with both continuous and discrete tradeoff assumptions.

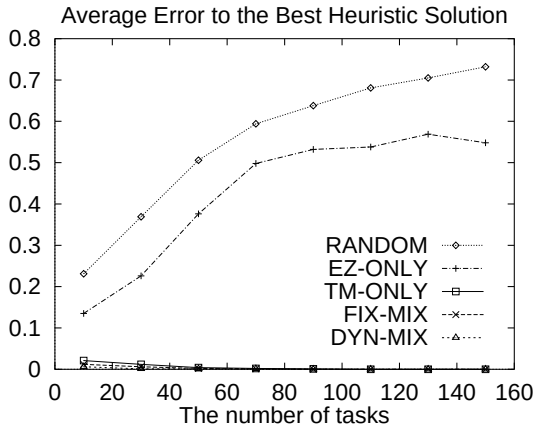

Fig. 6. Average error of heuristic solutions to the best heuristic solution as a function of the number of tasks ($|\tau_{sys}|$).

References

1. H. Aydin, R. Melhem, D. Mosse, and P.M. Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proc. of IEEE Real-Time Systems Symposium*, December 2001.
2. S. Furber. ARM system architecture. Addison Wisley, New York, NY, 1996.
3. M.R. Gaery and D.S. Johnson. Computers and interactability: a guide to theory of NP-completeness. W.H. Freeman and Company, 1979.
4. L. Goudge and S. Segars. Thumb: Reducing the cost of 32-bit RISC performance in portable and consumer applications. In *Proc. of the 1996 COMPCON*, September 1996.
5. J. Hennessy and D. Patterson. Computer architecuture - a quantitive approach. Morgan Kaufmann, 2nd edition, 1996.
6. I. Hong, G. Qu, M. Potkonjak, and M.B. Srivastava. Synthesis techniques for low-power hard real-time systems on variable voltage processors. In *Proc. of IEEE Real-Time Systems Symposium*, pages 178–187, 1998.
7. W. Kim, J. Kim, and S.L. Min. A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis. In *DATES*, 2002.
8. C. Rusu, R. Melhem, and D. Mosse. Maximizing the system value while satisfying time and energy constraints. In *Proc. of IEEE Real-Time Systems Symposium*, pages 246–255, December 2002.
9. I. Shin, I. Lee, and S. Min. Embedded system design framework for minimizing code size and guaranteeing real-time requirements. In *Proc. of IEEE Real-Time Systems Symposium*, pages 201–211, December 2002.
10. Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *Proc. of International Conference on Computer-Aided Design*, pages 365–368, 2000.
11. D. Sweetman. See MIPS Run. Morgan Kaufmann, San Francisco, CA, 1999.
12. F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. of IEEE Annual Foundations of Computer Science*, pages 374–382, 1995.