Throughput-Constrained Voltage and Frequency Scaling for Real-time Heterogeneous Multiprocessors

Pengcheng Huang\textsuperscript{1}, Orlando Moreira\textsuperscript{2}, Kees Goossens\textsuperscript{3}, Anca Molnos\textsuperscript{1}
\textsuperscript{1}Delft University of Technology, \textsuperscript{2}ST Ericsson, \textsuperscript{3}Eindhoven University of Technology
p.huang-1@student.tudelft.nl, orlando.moreira@stericsson.com, k.g.w.goossens@tue.nl, a.m.molnos@tudelft.nl

Abstract

Voltage and Frequency Scaling (VFS) has been shown to reduce energy consumption effectively on system level. Most existing work in this field focused on deadline-constrained applications with finite schedule lengths. However, in typical real-time streaming applications, data processing is repeatedly activated by indefinitely long data streams and operations on successive data instances are overlapped to achieve a tight throughput. Such behavior requires new VFS scheduling policies. In this paper, we solve throughput-constrained VFS problems for real-time streaming applications with discrete frequency levels on a heterogeneous multiprocessor platform. We propose different scaling algorithms for a multiprocessor platform with local voltage-frequency switches per processor, and for a multiprocessor platform with a global voltage-frequency switch for all processors. Moreover, we prove NP-hardness for the local VFS problem and maximal open complexity for the global VFS problem. We formulate the local voltage scaling problem using a Mixed Integer Linear Programming (MILP) formulation. For its global counterpart, we propose a three-stage heuristic incorporating MILP. Experiments on our modem benchmarks show that the discrete local VFS algorithm achieves energy savings close to its continuous counterpart, and local VFS is much more beneficial in terms of energy saving than global VFS. For example, for Wireless LAN (WLAN), compared to a policy that simply switches off the processors when not active, the continuous local VFS algorithm reduces energy consumption by 29\%, while the discrete local and global VFS algorithms reduce energy by 28\% and 16\%, respectively.

1 Introduction

In hard real-time streaming applications, data processing typically operates on indefinitely long data streams and the schedule overlaps [14] across successive iterations, i.e. operations from different iterations can execute simultaneously on different processors. For these applications, a primary temporal concern is the throughput requirement, which must be met along with any latency requirement.

The application domain includes Software-Defined Radio (SDR) [20], where the baseband processing activities of several radio standards run concurrently on an embedded multiprocessor. Typically, a baseband processing application is repeatedly activated by a periodic source from which it receives data and must keep up with its rate. Baseband processing can be divided into three main stages: filtering, (de)modulation and (de)coding. Iterations of processing are overlapped to meet a tight throughput requirement, e.g., filtering of a new data sample can happen simultaneously with demodulation of a previous data sample. Furthermore, cross-iteration data dependencies exist, that is, data produced by one iteration can be required for the processing of another iteration. For example, for many standards, the output of a decoder is fed back as input to its next iteration [7]. In another example, the result of the decoding stage for an iteration may be used to configure the demodulation stage of a subsequent iteration.

In addition to the real-time requirements, SDR applications also require low-energy processing, since they typically run on battery-operated devices. Voltage and Frequency Scaling (VFS) has already been demonstrated to be effective in reducing energy on system level by adjusting the system’s voltage and frequency levels [2]. From the application’s point of view, VFS changes the execution times of tasks and thus affects the schedule. For real-time applications, the main challenge becomes finding a schedule that makes use of VFS to save energy, but that still meets all timing requirements.

Using VFS for streaming applications is difficult for three reasons. First, since schedules are typically infinite, the problem size of VFS for streaming applications is infinite by nature. Second, because the schedule is overlapped, we must take into account data dependencies both within an iteration and across iterations [19]. Last, we must handle both throughput and latency constraints.
In this paper, we address the problem of performing throughput-constrained VFS for hard real-time streaming applications, with discrete frequency-voltage levels, running on a heterogeneous multiprocessor. Specifically, we propose compile-time VFS [2] techniques, where voltage and frequency settings are calculated off-line and then applied at run-time according to a pre-calculated schedule. We assume that the processors are switched off and consume no energy when not active. Furthermore, we neglect transition overheads when switching between different voltage and frequency points, and focus on addressing the unique features of streaming applications regarding VFS. However, the proposed techniques can be extended with existing techniques as proposed in [1] to further take into account transition overheads.

A considerable amount of work has been done on system level VFS. Yao et al. proposed in [22] the first DVS approach which can dynamically change the supply voltage over a continuous range. Andrei et al. [1] proposed an approach that optimally solves the voltage scaling problem for multi-processor systems with imposed time constraints. Their continuous voltage scaling is solved using convex nonlinear programming [16] with polynomial time complexity, while the discrete problem is proved strongly NP hard and is formulated as mixed integer linear programming problem. However, all these approaches focus on deadline-constrained VFS policies. To the best of our knowledge, the only VFS work existing in the streaming application domain [11] is proposed by Nelson et al. In their VFS algorithm, each task is assigned an individual frequency and voltage is scaled continuously. They formulate a convex programming problem with a throughput constraint. They then extend their continuous VFS algorithm to handle discrete frequency levels by rounding up the results to a limited set of discrete frequency levels. There are problems with this approach. First, their assumption that frequency is proportional to dynamic voltage while performing VFS is unrealistic. Second, their handling of discrete frequency levels by rounding up the results of a continuous distribution can lead to a severe loss of energy savings.

We use Single-Rate Data Flow (SRDF) [6] to model streaming applications. The SRDF model can be used to efficiently analyze the real-time behavior of a streaming application mapped onto a multiprocessor platform [8]. In this paper, we determine VFS policies by generating Static Periodic Schedules (SPS) [15], and a corresponding static periodic sequence of VFS operating points for each Voltage-Frequency Switch (VF-switch). We use SPS to reduce the size of the VFS problem, from infinite to finite. We know from the literature that, for SRDF, there is always a SPS that achieves the maximum attainable throughput of the graph [9], thus guaranteeing that opting for SPS does not hinder us from achieving our throughput requirement.

As the key contribution of this paper, we propose algorithms to perform VFS on streaming applications with infinite schedules, and hard real-time throughput requirements, assuming discrete VF-switches. We also show that (in Section 3), compared to adapt conventional deadline-constrained VFS techniques, it’s desirable to directly address throughput-constrained VFS problems by the proposed algorithms.

We consider VFS problems for both multiprocessors using local VF-switches and for multiprocessors using a single, global VF-switch. We also investigate the problem complexity for both VFS problem variants. We solve continuous throughput-constrained VFS by using more accurate models than those used in [11]. The solution is still formulated as a convex program. Although continuous VFS is not very realistic from a practical perspective, it does provide us with an upper bound to how much energy we can save with VFS, and allows us to benchmark our discrete approaches. The discrete local VFS problem is formulated as a mixed integer linear program, while its global counterpart is solved by a three-stage heuristic that incorporates mixed integer linear programming.

The remainder of the paper is organized as follows: Section 2 provides preliminaries regarding SRDF, power and delay models, and the target multiprocessor platform. Section 3 presents an example to illustrate the throughput-constrained VFS problem. Section 4 introduces the different variants of the throughput-constrained VFS this paper addresses. Sections 5 and 6 provide solutions for continuous and discrete VFS problems with local VF-switches. This is followed by a description of our approach to the discrete VFS problem with a global VF-switch in Section 7. Section 8 reports experimental results and analyses them. Section 9 concludes the paper.

2 Preliminaries

In this section, we present our target platform, the Single Rate Data Flow model we used to model real-time streaming applications, its timing analysis properties, and the power model.

2.1 Target Platform

Heterogeneous multi-processor platforms provide a good trade-off between flexibility, performance and cost [13]. In this paper we consider embedded systems which are realized as heterogeneous Multi-processor System on Chip (MPSoC), where all processors (general processing cores and application-specific cores) have their own local memories and are connected together with peripherals (such as I/O ports) via a Network-On-Chip (NoC) [3].
For our modem example, the platform has three types of processors, the Embedded Vector Processor (EVP) [20], the Software Decoder Processor and the ARM processor, each of which may have multiple instances, as shown in Fig. 1.

![Target platform](image)

**Figure 1. Target platform**

Formally, the multiprocessor platform $\prod (\Pi, F, p)$ has the processor set $\Pi = \{ \pi_0, \pi_1, \ldots, \pi_m \}$, where $\pi_i$ is the $i$th processor in the platform. A set of frequency levels $F$ is available to all processors. The valuation $p$ stands for power dissipation associated with each frequency level for each processor $p : \Pi \times F \rightarrow \mathbb{R}_0^+$. Frequencies and voltages of all processors can be managed locally (by local VF-switches per processor) or globally (by a single global VF-switch).

We investigate VFS algorithms for both cases.

### 2.2 Data Flow Model

#### 2.2.1 SRDF graph

To model real-time streaming applications, we use SRDF [6]. A SRDF application graph $G = (V, E, d, t)$ is a directed graph in which the input and output rates of each node can be specified a priori. Nodes (actors) $V$ represent the set of actual tasks and edges $E$ represent the set of communication channels between different nodes. Data is transported in discrete chunks, which are referred to as tokens. A node is enabled by the availability of input data and FIFO space for output data. An enabled node can be fired. For each node $i \in V$, it fires by consuming from its incoming edges and producing on its outgoing edges the same amount of tokens. Each node input and output is independent of the node state and node firings are free from side effects. An iteration of the SRDF corresponds to one firing of every actor in the SRDF. Nodes in a SRDF graph are executed essentially infinitely often.

The valuation of edges $d : E \rightarrow \mathbb{N}_0$ represents the initial token distribution on edges. For example, $d(i, j)$ represents the initial tokens on edge $(i, j)$, which we also refer to as delay on edge. A SRDF is timed when the execution time of each actor is given by the valuation $t : V \rightarrow \mathbb{R}_0^+$.

In SRDF, an edge $(i, j)$ together with the delay $d(i, j)$ on it represent the precedence constraint [19], which can be formalized as follows:

$$s(j, k) \geq e(i, k - d(i, j)), \forall k \geq d(i, j)$$

(1)

where $s(j, k)$ and $e(i, k - d(i, j))$ represent the $k$th start time of node $j$ and $(k - d(i, j))$th end time of node $i$, respectively.

Given a timed SRDF, the throughput upper-bound of the system can be computed by the inverse of the Maximum Cycle Mean (MCM) [15] of the SRDF, which is defined as

$$\max_{e \in C(G)} \left( \frac{\sum_{i \in V(c)} t(i)}{\sum_{e \in E(c)} d(e)} \right)$$

(2)

where $C(G)$ is the set of cycles in the SRDF graph, $V(c)$ represents the set of actors and $E(c)$ represents the set of edges, both on cycle $c$. The execution time of node $i$ is represented as $t(i)$.

Furthermore, deadline constraints can be explicitly expressed in terms of the throughput constraint in a SRDF model [9]. Hence in this paper, without losing generality, we will only consider the throughput constraint.

#### 2.2.2 Static periodic scheduling for SRDF

To reduce the VFS problem size imposed by infinite schedules, we restrict ourselves to static periodic scheduling in this paper, which can be specified by a triple:

$$\text{SPS} = \{ \pi(i), s(i), \mu_d \}, \forall i \in V$$

(3)

where $\pi(i)$ and $s(i)$ represent the processor to which actor $i$ is mapped and the start time of actor $i$, respectively. $\mu_d$ stands for the SPS period. For static periodic scheduling, strict periodicity is enforced when schedule task executions [15]:

$$s(i, k) = s(i, 0) + \mu_d \times k, \forall i \in V$$

(4)

Due to this periodicity, we have a finite representation for an infinite schedule, and furthermore, a minimum throughput requirement (e.g. $\mu_d^{-1}$) can be explicitly converted into a maximum period requirement (e.g. $\mu_d$).

A SPS period $\mu_d$ for a SRDF is admissible only when $\mu_d \geq \text{MCM}$ [10]. Recall Equation 1, for SPS, the precedence constraint imposed by edge $(i, j)$ becomes:

$$s(j, k) - e(i, k) \geq -\mu_d \times d(i, j), \forall k \geq d(i, j)$$

(5)

Given a SRDF graph, the precedence relations introduced by a SPS can be represented as additional edges $E_S$ in the graph, and $E_S \cup E$ essentially implies the static ordering of task firing sequences after scheduling. An example of a statically scheduled SRDF and its timing diagram are shown in Fig. 2. The task graph (Fig. 2(a)) is scheduled
onto two processors \( \pi_1 \) and \( \pi_2 \) and the scheduled SRDF (Fig. 2(b)) is constructed, in which dotted edges represent \( E_S \). Fig. 2(c) shows the overlapped periodic scheduling diagram. Bullets(●) in graphs represent delays on edges.

![Task graph](image)

![Scheduled task graph](image)

![Static periodic scheduling diagram](image)

**Figure 2. SPS for SRDF**

### 2.3 Power And Delay Models

We only consider dynamic power in this paper, for which the standard model [2] is used:

\[
P_{\text{dynamic}} = \alpha C v_{dd}^2 f
\]

(6)

where \( \alpha \) models the circuit switching activity, \( C \) represents the switched capacitance and \( v_{dd} \) and \( f \) represent the voltage level and frequency level, respectively. The voltage and frequency pair \((v_{dd}, f)\) determines the execution mode of a processor. When performing VFS, \( v_{dd} \) scales with \( f \) [2], which is given by

\[
f = K \cdot \frac{(v_{dd} - v_{th})^\delta}{v_{dd}}
\]

(7)

where \( v_{th} \) is the switching threshold voltage of a transistor, \( \delta \) reflects the charge velocity saturation imposed by the technology, and \( K \) represents a circuit dependent constant.

### 3 Motivational Example

To demonstrate the influence of infinite overlapped scheduling and cross-iteration data dependences on voltage and frequency scaling, consider the following example. For clarity reasons, we assume throughput-constrained VFS with discrete frequency levels here. The hardware platform in the example has three processors \( \pi_1, \pi_2 \) and \( \pi_3 \), with detailed description depicted in Table 1. Two voltage and frequency levels are available to all three processors. Power dissipations are made integer for simplicity reasons. The virtual processor \( \pi_{virtual} \) in the example platform is used to map external non-scalable source tasks, and only the higher frequency is available on it. The processors need to execute tasks \((S, A, B \text{ and } C)\) with precedence relations shown in Fig. 3(a). Task \( S \) is the external source task and the other three are processing tasks. All tasks initially execute in the high frequency mode. Fig. 3(b) shows the SPS diagram before VFS. We convert the minimum throughput requirement to a maximum period requirement of 40\(\mu s\). The total energy consumption per period of this schedule is the sum up of energy dissipation in each frequency level of all tasks \(E_{total} = 2 + 3.2 + 0.6 = 5.8\mu J\) (processors are switched off and consume no energy when not active).

<table>
<thead>
<tr>
<th>Processor</th>
<th>Voltage(V)</th>
<th>Frequency(MHZ)</th>
<th>Power(mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \pi_1 )</td>
<td>1.1</td>
<td>312</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>156</td>
<td>60</td>
</tr>
<tr>
<td>( \pi_2 )</td>
<td>1.1</td>
<td>312</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>156</td>
<td>50</td>
</tr>
<tr>
<td>( \pi_3 )</td>
<td>1.1</td>
<td>312</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>156</td>
<td>7</td>
</tr>
<tr>
<td>( \pi_{virtual} )</td>
<td>1.1</td>
<td>312</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 3. Application Example**

As mentioned, schedules for real-time streaming applications are infinite in terms of schedule length and are typically overlapped across successive iterations, which prevents traditional deadline-constrained VFS policies [1] from being directly used. The reason is that they only aim at one iteration of task executions. We propose here a solution to get over this, which is by splitting the infinite SPS into minimum periodic regions (blocked schedules) and then applying deadline-constrained VFS policies to such a blocked schedule. The original period constraint can be converted into a deadline constraint for the blocked schedule. In Fig. 4, we show how to adapt a deadline-constrained discrete VFS algorithm proposed in [1] for the example shown in Fig. 3. First, since a blocked schedule is not available from the original schedule, the SRDF graph (Fig. 3(a)) is first retimed\(^1\) to get a blocked schedule and then executed.

\(^1\)Retiming [25] is used to redistribute the original tokens in a SRDF and to improve blocked schedules.
in a periodic fashion (Fig. 4(a)). Furthermore, as precedence relations for such a blocked schedule can be rearranged within each block, a Direct Acyclic Graph (DAG) (with zero execution time Begin and End nodes) is constructed from the original SRDF (for the minimum periodic region 80\(\mu\)s \(\sim\) 120\(\mu\)s in Fig. 4(a)). This DAG is then used as the input for voltage scaling with a deadline constraint of 40\(\mu\)s. As the results, task A and task C extend their execution times to 20\(\mu\)s and 40\(\mu\)s, respectively. Task C is not scaled. The total energy consumption for one period after scaling is \(E'_{total} = 1.2 + 3.2 + 0.54 = 4.94\mu J\).

The motivational example shows that by directly addressing the throughput-constrained VFS problem, we get extra energy savings of \(\frac{4.94\mu J - 4.4\mu J}{4.4\mu J} = 9.3\%\) compared to an adapted deadline-constrained VFS algorithm. The fundamental reason is that in any valid schedule (e.g. a SPS) for streaming applications, static slack exists across iterations due to the cross-iteration precedence constraints. Though we can unfold\(^2\) the SRDF to let a deadline-constrained VFS approach handle schedules with multiple iterations to explore more static slack, the throughput-constrained VFS problem cannot be solved optimally in this way since cross-iteration static slack still exists even for the unfolded task graph. Hence, despite the high complexity of applying a deadline-constrained VFS algorithm, directly addressing throughput-constrained VFS problems is desirable.

4 Problem Formulation and Variants

To perform VFS for real-time streaming applications, we assume mapping and static ordering of tasks are performed by a scheduling flow as proposed in [8]. The output of the different VFS variants proposed in the paper are static periodic schedules with corresponding static periodic sequences of voltage and frequency points for each VF-switch. We hereby first formalize the throughput-constrained VFS problem.

**Definition 4.1** A throughput-constrained VFS problem (TC-VFS): A SRDF \(G(V, E, d, t)\) with throughput requirement \(\mu_d^{-1}\) and a heterogeneous multiprocessor platform \(\Pi(V,F,p)\) are given. The hardware platform can be a multi-clock domain platform with local VF-switches or a single clock domain platform with a global VF-switch. The frequency levels \(F\) can have any arbitrary value in a fixed range or just have a limited number of discrete values. All task execution times are evaluated for the base frequency \(f_b \in F\). For each task \(i \in V\), the number of clock cycles can be calculated as \(nc(i) = t(i) \times f_b\). A set of external source tasks \(V_S \subset V\) are not scalable in terms of execution time, and a subset the processors \(\Pi_S \subset \Pi\) (used to map the external sources) are assumed to consume zero power. Task mapping is represented as \(\pi : V \rightarrow \Pi\) and static ordering represented as \(\sigma : \Pi \rightarrow \alpha^n\), where \(\alpha^n = [i_1, i_2, \ldots, i_n]\) is the set of actor firing sequences of a processor, and \(i_1, i_2, \ldots, i_n \in V\). Both task mapping and ordering are given a priori. The problem is to find a static periodic schedule with scaled task execution times using VFS, such that the total energy consumption is minimized under the throughput requirement. In other words, the output of VFS is a distribution of clock cycles to the voltage and frequency levels for all tasks \(V \times F \rightarrow \Pi\), and a static periodic schedule \(\{\pi(i), s(i), \mu_d\}\) with scaled task execution times fulfilling the throughput requirement \(\mu_d\).

\(^2\)The unfolding strategy [14] schedules \(N\) iterations of a SRDF together, which often leads to improved blocked schedules.
According to the availability of VF-switches supported, we consider the following three variants of TC-VFS problems in this paper:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Frequency</th>
<th>VF-switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC-CLVFS</td>
<td>Continuous</td>
<td>Local</td>
</tr>
<tr>
<td>TC-DLVFS</td>
<td>Discrete</td>
<td>Local</td>
</tr>
<tr>
<td>TC-DGVFS</td>
<td>Discrete</td>
<td>Global</td>
</tr>
</tbody>
</table>

A convex programming problem is formulated for the TC-CLVFS problem while for the TC-DLVFS problem we formulate it as a mixed integer linear program and for the TC-DGVFS problem we propose a three-stage heuristic.

5 Continuous Local VFS

Continuous VFS with local VF-switches for throughput-constrained streaming applications was proposed in [11], where the authors formulate it as a convex programming problem. However, they assume that frequency is proportional to dynamic voltage, which is unrealistic. In our approach presented in this section, we use the frequency and dynamic voltage relation as stated in Equation 7 and restrict the frequency due to limits of the dynamic voltages of the system. The VFS problem can still be formulated as a convex program, and an optimal solution can be found by a polynomial time complexity algorithm [12].

5.1 Problem Definition and Algorithm

**Definition 5.1** A throughput-constrained continuous local VFS problem (TC-CLVFS) is a TC-VFS problem where the frequency set can have any arbitrary frequency level in a fixed range and each processor has its own local VF-switch; in other words, frequency can be scaled continuously and independently.

The goal of the following TC-CLVFS formulation is to minimize the total dynamic energy consumption by finding the optimal executing frequency for each task. \( \pi(i) \) represents the mapped processor of task \( i \) while \( f(i) \) and \( v(i) \) represent its working frequency and dynamic voltage level, respectively. Function \( p(\pi(i), f(i)) \) indicates the power dissipation of task \( i \) running with frequency \( f(i) \) on the mapped processor \( \pi(i) \). For each task \( i \), the number of clock cycles \( nc(i) \) remains unchanged while performing VFS [17]. The execution time of task \( i \) is represented as \( \frac{nc(i)}{f(i)} \). The optimization has to comply to the precedence constraints, the voltage-frequency relations and the constraints on the supported voltages for each processor on the system. For the non-scalable tasks \( V_S \), frequency level is equal to the base frequency. Since the voltage and frequency relation we use here is a convex function, the formulated problem is still a convex programming problem.

**TC-CLVFS algorithm**

minimize \( \sum_{i \in V} p(\pi(i), f(i)) \times \frac{nc(i)}{f(i)} \)

subject to :

\[
\forall (i, j) \in E \cup E_S, \quad s(j) - s(i) \geq -d(i, j) \times \mu_d + \frac{nc(i)}{f(i)}
\]

\[
\forall i \in V, \quad \frac{1}{f(i)} = K \cdot (v_{dd}(i) - v_{dd\min}(\pi(i)))^\delta
\]

\[
\forall i \in V_S, \quad f(i) = f_b
\]

Notice that in the above formulation, only task mapping and ordering (represented as \( E \cup E_S \)) from a static schedule are enforced, which means that the formulation finds a new SPS by assigning the start times \( \{s(i)\}_{\forall i \in V} \) and execution times \( \{\frac{nc(i)}{f(i)}\}_{\forall i \in V} \) to all tasks under precedence constraints as shown in Equation 5.

6 Discrete Local VFS

In the previous section, we showed how continuous local VFS can be formulated as a convex programming problem, which can be optimally solved in polynomial time. This provides a theoretical lower bound on possible energy savings. However, real platforms can only support a limited number of discrete frequency and voltage levels. In [11], the authors round up continuous VFS results to a set of discrete frequency levels in a real platform, but without guaranteeing optimality. In this section we first investigate the problem complexity of the throughput-constrained discrete VFS problem with local VF-switches. We further optimally solve this variant of the TC-VFS problem by formulating it into a MILP program.

6.1 Problem Definition and Algorithm

**Definition 6.1** A throughput-constrained discrete local VFS problem (TC-DLVFS) is a TC-VFS problem where the frequency set can only have a limited number of discrete values and each processor can scale operating frequency independently.

**Theorem 6.1** The TC-DLVFS problem given in Definition 6.1 is NP-hard.

**Proof Sketch** The Deadline-Constrained Discrete Local VFS (DC-DLVFS) problem is known to be NP-hard [1]. By showing that DC-DLVFS problem can be reduced to a subset of our proposed TC-DLVFS problem in polynomial time, NP-hardness is proved.
In the following, we give a mixed integer linear programming (MILP) formulation for the TC-DLVFS problem. The goal is to optimally assign the number of clock cycles of each task to each frequency level, which consequently minimizes the total energy consumption. Energy consumption in this case is the sum of energy dissipations associated with all task-frequency pairs. We use the variable \( nc(i, f) \) to represent the number of clock cycles task \( i \) spends on frequency level \( f (f \in F) \). Again precedence constraints must be respected. The number of clock cycles variable \( nc(i, f) \) must be integers and the total number of clock cycles of each task is a constant. For the non-scalable tasks, execution can only be done on the base frequency \( f_b \). Since all the constraints are linear, the formulated problem is a MILP problem. MILP problems are in general NP-complete [18] and require non-polynomial time to solve. However by dropping the constraints that variable \( nc(i, f) \) be integers, we can get a tight approximation algorithm represented as a linear programming problem, which can be optimally solved in polynomial time.

### 7 Discrete Global VFS

In the previous two sections, we solved the throughput-constrained local VFS problems for both continuous and discrete cases. Those local policies assume that the platform has local VF-switches per processor. However, it is not addressed how beneficial in terms of energy saving by having local VF-switches compared to a single VF-switch for the entire platform. In this section, we propose a three-stage heuristic for such global VFS problem.

Concerning VFS with a global VF-switch, tasks running at the same time on different processors are forced to switch frequency simultaneously. The crucial observation here is that we are scaling parallel blocks from a coarser granularity point of view. By a parallel block, we mean the shortest time interval of a schedule within which tasks remain the same. For such a parallel block, a higher power dissipation means a higher priority to stretch execution in order to save energy. Consequently, the search for a VFS solution can be done in three stages: (1) find a blocked schedule with as short as possible schedule length; (2) identify the parallel blocks; (3) perform VFS for the parallel blocks found. The goal for the first stage is to get as much as possible global static slack at the end of a blocked schedule, which can be used to stretch task executions. The second stage identifies the parallel blocks and further calculates power and time characteristics of them, while in the third stage, parallel blocks extend their execution times under the original throughput constraint to save energy.

As an illustrative example, recall the motivational example in Section 3. Our proposed global discrete VFS algorithm works as follows: first, the original SPS schedule as shown in Fig. 3 is compacted by using a source with shorter period. The compact schedule (Fig. 6(a)) is then splitted into minimum periodic regions. Such a periodic region (Fig. 6(b)) is further divided into parallel blocks. The identified parallel blocks are then scaled as if they are executing on a single processor with a deadline constraint (40\( \mu s \)) converted from the original throughput requirement. Since the source task is non-scalable, the parallel blocks are scaled in such a way that after scaling, task \( S \) has the same execution time as in the original application graph (40\( \mu s \) for our example). Fig. 6(c) shows the scaling results: \( PB_1 \) extends its execution to 20\( \mu s \) and \( PB_2 \) is not scaled. The reduced energy is \( E'' = 1.2 + 2.6 + 0.6 = 4.4\mu J \).

In the remainder of this section, we investigate the problem complexity for the TC-DGVFS problem. Furthermore, we formalize our proposed three-stage heuristic for such global VFS problem.
7.1 Problem Definition and Algorithm

Definition 7.1 A throughput-constrained discrete global VFS problem (TC-DGVFS): A TC-DGVFS problem is a TC-VFS problem where the frequency set can only have a limited number of discrete values and all the tasks running at the same time on different processors have the same operating frequency, namely, tasks must scale frequency identically and simultaneously.

Theorem 7.1 The TC-DGVFS problem given in Definition 7.1 is maximal open.

Proof Sketch The single machine discrete time-cost scheduling problem [4] is maximal open and can be reduced to subset of the TC-DGVFS problem, where mapping and ordering are unknown. Hence Theorem 7.1 holds.

The proposed heuristic for the TC-DGVFS problem has three main stages, as shown below:

<table>
<thead>
<tr>
<th>Pseudo-code: TC-DGVFS algorithm</th>
</tr>
</thead>
</table>

- Get a compact SPS  
- Let $G'$ denote a copy of the original graph $G$, where $t(i) = 0, \forall i \in V_S \in G'$

1: While $MCM(G) \geq MCM(G')$ do  
   For each $i \in V_S$  
   $t(i) = \frac{t(i)}{\Delta t}$ (SD $\geq$ 1)  
   Endfor  
   SD++  
Endwhile

2: Get a new SPS for the updated graph $G$

- Identify the set of parallel blocks PB  
3: For any time $t \in BS$ (blocked schedule from the SPS)  
   If Parallelism at $t \neq$ parallelism at $t + \Delta t$ ($\Delta t \to 0$)  
      Register parallel block at $t$ in PB  
   Endif  
   Endfor

4: For each parallel block $j \in PB$ (parallel block set)  
   For each frequency level $f \in F$  
      - Power dissipation of all tasks $i \in j$  
      $p_{\rho}(i, f) = \sum_{i,j} p(i, f)$  
   Endfor

Endfor

- VFS for identified parallel blocks from a BS  
5: VFS for PB, formulated as MILP program

As discussed, we need to first find a blocked schedule of parallel blocks with as much as possible global static slack at the end. This is done by getting a compact SPS, as shown in the first stage of the algorithm. In detail, step 1 tries to shrink the sources such that the new graph achieves the lower bound MCM of the SRDF $G'$ consisting of only processing tasks. A compact SPS is retrieved in step 2. Identifying a blocked schedule from the compact SPS is done through step 3 to step 4, which set up the timing and power characteristics of the identified parallel blocks.

The goal of step 5 is to find a VFS solution for the identified parallel blocks within a single blocked schedule. We formulate it into the following MILP program.

VFS-stage algorithm (step 5 in TC-DGVFS algorithm)

\[
\text{minimize} \sum_{i \in PB} \sum_{f \in F} p_{\rho}(i, f) \times \frac{nc(i, f)}{f}
\]

subject to:

\[
\sum_{i \in PB} \sum_{f \in F} \frac{nc(i, f)}{f} \leq \mu_d
\]

\[
\forall c \in V_S, \forall i \in PB_c, \sum_{f \in F} \frac{nc(i, f)}{f} = \frac{nc(c)}{f_b}
\]

\[
\forall i \in PB, \forall f \in F, nc(i, f) \in \mathbb{N}
\]

In the above formulation, $nc(i, f)$ represents the number of clock cycles parallel block $i$ spent on frequency $f$, while $p_{\rho}(i, f)$ represents its power dissipation on frequency $f$. The insight into the VFS stage is that it is identical to perform VFS for chained applications on a single processor. Since sources are non-scaleable, the set of different parallel blocks with the same spread source $\varsigma$, $PB_c$, must sum up to the original execution time of source $\varsigma$ after scaling.

8 Experiments and Results

In this section, we demonstrate the applicability of the presented VFS algorithms by conducting a set of experiments with several real life streaming applications in the domain of Software-Defined Radio (WLAN, TDSCDMA, AM Radio, ChannelEq). The real-time radios are scheduled on to a modem platform composed of 3 ~ 9 processors. The technology dependent parameters of the processors are considered to correspond to a CMOS fabrication in 45 nm. The processors are switched off when not active, and energy consumptions are compared to when no VFS is applied.

We first validate the applicability of our proposed VFS techniques on the set of radio applications. We summarize the results in Table 3. In particular, we run the 3 variants of our proposed VFS algorithms, where the discrete frequency level local and global algorithms use 5 frequency levels to perform VFS. The set of frequency levels is a geometric series with a common ratio of 2.

For our WLAN and TDSCDMA applications, the results show that energy consumptions are significantly reduced by the VFS algorithms proposed. For example, for the WLAN application, the TC-CLVFS algorithm reduces...
Table 3. Results for radio applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Energy Reduction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TC-CLVFS</td>
</tr>
<tr>
<td>WLAN</td>
<td>29</td>
</tr>
<tr>
<td>TDSCDMA</td>
<td>37</td>
</tr>
<tr>
<td>AM Radio</td>
<td>10</td>
</tr>
<tr>
<td>ChannelEq</td>
<td>15</td>
</tr>
</tbody>
</table>

energy by 29%, while the TC-DLVFS algorithm and the TC-DGVFS algorithm reduce energy by 28% and 16%, respectively. The energy reductions for these two applications are due to large amount of static slack. Furthermore, the results of the AM Radio and the ChannelEq applications show that, for applications with tight schedules (small amount of static slack), energy saving differences of the VFS algorithms can be significant. For the AM Radio application, the TC-CLVFS algorithm saves 10% energy, while the TC-DLVFS and TC-DGVFS algorithms save energy by 6% and 1%, respectively. In case that the static slack amount is small, the continuous algorithm goes through a stage when energy reduces dramatically due to the convex relation between power and frequency, while the discrete local algorithm has to assign most of the clock cycles of an application to the highest frequency level, which saves energy poorly. In addition to this, the performance of our discrete global algorithm further depends on how much global static slack exist in a compact schedule, as described in Section 7. Since the AM Radio works mainly in a chained fashion under a tight schedule, the discrete global algorithm can not efficiently take use of the static slack compared to the discrete local algorithm. Thus, having local VF-switches is desirable in terms of energy saving.

The second set of experiments are performed to compare the energy saving abilities of our discrete local VFS algorithm with the VFS technique proposed in [11], which incorporates discrete frequency levels by rounding up VFS results from a continuous VFS algorithm. In such a rounding up heuristic, the calculated frequency level of a task is conservatively rounded up to its closest frequency level. We give the comparison results for the WLAN and TDSCDMA applications in Table 4. As discussed, the way to handle discrete frequency levels by rounding up can lead to severe loss of energy savings. This is because static slack is generated when rounding up frequency values to make the tasks non-necessarily faster. On the other hand, directly addressing discrete frequency levels can lead to optimal VFS solutions, as described in our TC-DLVFS algorithm in Section 6. For the WLAN application, the rounding up heuristic almost saves no energy (1%) while our optimal TC-DLVFS algorithm reduces energy by 28%. This is because the VFS results for the WLAN application by the the TC-CLVFS algorithm are almost all caught between the highest two frequency levels. By rounding up, the continuous VFS results are almost all rounded up to the highest frequency, which saves almost no energy. For the TDSCDMA application, the rounding up heuristic and our TC-DLVFS algorithm save energy consumption by 33% and 36%, respectively. We observe for this set of experiments that, while the rounding up heuristic can not produce optimal VFS solutions, there exist extreme cases (e.g. the WLAN application) when this heuristic loses a great potential in energy savings. Hence, directly addressing discrete frequency levels by our proposed TC-DLVFS algorithm is desirable.

We further evaluate the effects of the number of discrete frequency levels on the proposed discrete local and global VFS algorithms. In particular, we compare the discrete algorithms taking utilization of 1 to 5 frequency levels, where 1 frequency level means no VFS. Fig. 8 presents the results for the WLAN and TDSCDMA applications. We can observe that energy consumption reduces dramatically from 1 to 3 frequency levels, and continues to reduce slightly till 5 frequency levels. This is due to the convex relation between energy consumption and frequency [23]. In particular, energy consumption increases dramatically fast when frequency is high, thus when scaling down the operating frequency of a task, the majority of energy savings are done through the initial stages of scaling down frequency.
detailed comparisons for the energy reduction differences have been discussed in the above experiments.

9 Conclusion

Throughput-constrained VFS problems are different from conventional deadline-constrained VFS problems in that they have to deal with infinite schedules and precedence constraints between successive iterations of tasks. In this paper, we investigated throughput-constrained VFS variants, evaluating the impacts of continuous or discrete frequency levels, local VF-switches per processor or a global VF-switch for the entire platform. We demonstrated that convex programming and mixed integer linear programming formulations can be used to solve those problems. We also proved NP-hardness for the discrete local VFS problem and maximal open complexity for the discrete global VFS problem. A number of experiments conducted on real life software-defined radios showed that our proposed VFS algorithms can achieve energy savings ranging from 10% to 40%.

References