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

Pengcheng Huang
Swiss Federal Institute of Technology (ETH) Zurich
Zurich, Switzerland
pengcheng.huang@tik.ee.ethz.ch

Orlando Moreira
ST-Ericsson
Eindhoven, The Netherlands
orlando.moreira@stericsson.com

Kees Goossens
Technical University of Eindhoven
Eindhoven, The Netherlands
k.g.w.goossens@tue.nl

Anca Molnos
Delft University of Technology
Delft, The Netherlands
a.m.molnos@tudelft.nl

ABSTRACT

Voltage and Frequency Scaling (VFS) can effectively reduce energy consumption at system level. Most work in this field has focused on deadline-constrained applications with finite schedule lengths. However, in typical real-time streaming, processing is repeatedly activated by indefinitely long data streams and operations on successive data instances are overlapped to achieve a tight throughput. A particular application domain where such characteristics co-exist with stringent energy consumption constraints is baseband processing. Such behavior requires new VFS scheduling policies. This paper addresses throughput-constrained VFS problems for real-time streaming with discrete frequency levels on a heterogeneous multiprocessor.

We propose scaling algorithms for two platform types: with dedicated VFS switches per processor, and with a single, global VFS switch. We formulate Local VFS using Mixed Integer Linear Programming (MILP). For the global variant, we propose a 3-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 more effective than global VFS. As an example, for a WLAN receiver, running on a modem realized as a heterogeneous multiprocessor, the continuous local VFS algorithm reduces energy consumption by 29%, while the discrete local and global algorithms reduce energy by 28% and 16%, respectively, when compared to a on/off energy saving policy.

1. INTRODUCTION

Hard real-time streaming applications operate on indefinitely long data streams and the schedule overlaps [12] across successive iterations, i.e. operations from different iterations execute simultaneously on different processors. A primary real-time concern is the *throughput* requirement.

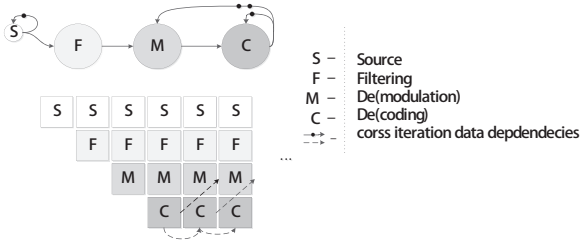

Figure 1: Simplified baseband processing.

An example is Software-Defined Radio (SDR) [17] , where baseband processing tasks run concurrently on an embedded multiprocessor. The radio is repeatedly activated by a periodic source and must keep up with its rate. Radios have three main stages: filtering, (de)modulation and (de)coding, as shown in Fig. 1 [9]. Iterations are overlapped to meet a tight throughput requirement, e.g., filtering of a new sample happens simultaneously with demodulation of a previous sample. Furthermore, cross-iteration data dependencies exist, i.e., data produced by one iteration can be required for the processing of another iteration.

In addition to the real-time requirements, transceivers require low-energy processing, as they typically run on battery-operated devices. *Voltage and Frequency Scaling* (VFS) has been shown to be effective in reducing energy at system level by adjusting voltage and frequency [5]. However, VFS changes the execution times of tasks and thus affects the schedule. For transceivers, the challenge is to find a schedule that makes use of VFS to save energy, while meeting strict timing requirements.

There are three difficulties in performing VFS on streaming applications. First, since schedules are typically infinite, the problem size is by nature infinite. Second, because the schedule is overlapped, we must account for data dependencies both within and across iterations [16]. Last, we must handle both throughput and latency constraints.

In this paper, we address throughput-constrained VFS for hard real-time streaming applications, with discrete voltage-frequency levels, running on a heterogeneous multiprocessor.

SAC'13 March 18-22, 2013, Coimbra, Portugal.

We propose compile-time VFS [5] techniques, where settings are calculated off-line and applied at run-time according to a pre-calculated schedule. We neglect transition overheads when switching between voltage/frequency points, and focus on the unique features of streaming applications. Ignoring the transition overhead underestimates energy consumption, leading to sub-optimality. However, for our reference hardware, and for our target applications, this underestimation is low. On our reference modem, switching voltage-frequency levels from maximum to minimum takes $20ns$. This is due to voltage ramping up/down, as frequency changes instantly by switching PLLs. This means that, pwe transition, the processor runs $20ns$ at the low frequency, while consuming energy at a higher voltage. In our experiments, all algorithms generated solutions with a low number of transitions per task. (in fact no more than two transitions in a single task were used, and a typical transceiver has between 9 and 20 tasks). Even pessimistically assuming consumption at maximum voltage during the transition, and two transitions per task, the total additional energy consumption per transition would be lower than 1% of the total energy consumption for our TDSCDMA receiver, and below 5% for our WLAN receiver, which has a particularly low absolute energy consumption per iteration when compared with other standards, due to its much faster rate. Moreover, our techniques can be extended with previous work [4] to account for transitions, at the cost of extra computational complexity.

We also neglect communication. However, our techniques can be extended to address it by adding communication tasks to the application graph. However, power models for such tasks must be provided.

We use *Single-Rate Data Flow* (SRDF) [7] to model streaming applications. SRDF can efficiently model and analyze the real-time behavior of a streaming application mapped onto a multiprocessor platform [9]. We determine VFS polices by generating *Static Periodic Schedules* (SPS) [9], and a corresponding static periodic sequence of VFS operating points for each *Voltage-Frequency Switch* (VF-switch). We use SPS to reduce the problem from infinite to finite. For an SRDF graph, there is always a SPS that achieves the maximum attainable throughput [9], thus guaranteeing that opting for SPS does not hinder us from achieving our throughput requirement. Cross-iteration data dependencies are modeled in SRDF by edges with delays [9]. Latency constraints can be modeled as additional precedence constraints on a throughput-constrained SRDF, as described in [9]. Thus, for the remainder of the paper we will talk only of the throughput constraint, but the reader should keep in mind that some of our benchmarks, such as WLAN, have stringent latency constraints that our solution meets, usign the latency modeling technique.

As the key contribution of this paper, we propose algorithms to perform VFS on streaming applications with infinite schedules, cross-iteration data dependencies, and hard latency and throughput requirements, assuming discrete VF-switches. We also show that it is desirable to directly address throughput-constrained VFS problems instead of adapting deadline-constrained VFS techniques.

We consider VFS problems for both multiprocessors using local VF-switches and a single, global VF-switch. This also allows us to compare energy savings between different hardware choices. We investigate the problem complexity of both variants. We solve continuous throughput-constrained VFS by formulating it as a convex program. Continuous VFS is not practical, but it provides an upper bound to the savings that can be obtained by VFS. The Discrete Local VFS problem is formulated as a Mixed Integer Linear Program (MILP), while its global counterpart is solved by a 3-stage heuristic incorporating MILP.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 provides preliminaries regarding SRDF, power modeling, and the target multiprocessor. Section 4 presents an example to illustrate the throughput-constrained VFS problem. Section 5 formalizes the throughput-constrained VFS problem. Sections 6 and 7 address respectively the continuous and discrete VFS variants with local VF-switches. Section 8 addresses discrete VFS with a global VF-switch. Section 9 reports and discusses our experiments. Section 10 concludes the paper.

2. RELATED WORK

There is a considerable amount of work on system level VFS. Yao et al. [20] proposed the first VFS approach which dynamically changes supply voltage over a continuous range. Andrei et al. [4] proposed an approach that optimally solves the VFS problem for multi-processors with imposed time constraints. They solve continuous VFS with a (polynomial complexity) convex program [13], prove the discrete problem is strongly NP hard and formulate it as a MILP.

Only a few VFS techniques have been proposed for streaming applications. Xu *et al* [19] considers task mapping and scheduling, by explicitly constructing pipeline stages from the task graph and then performing VFS on all pipeline stages. This approach cannot address general deadline requirements, and optimality cannot be guaranteed for general task graphs. Liu *et al* [8] explicitly retime an acyclic task graph to construct loop kernels and then apply VFS to the loop kernel with the throughput constraint converted to a corresponding deadline constraint. It is not clear whether optimality of energy saving is lost with retiming. Moreover, this approach cannot optimally handle cross-iteration data precedences. Wang *et al* [18] propose a two stage approach to perform VFS for streaming applications. First, they use the same graph transformation as proposed in [8]. They then apply genetic algorithms to search for solutions. Their approach suffers from the same limitations as [8]. In conclusion, none of the existing work handles general constraints and cross-iteration data dependencies optimally.

The technique most similar to ours is proposed in [10], where the VFS algorithm assigns a frequency to each task, and voltage is scaled continuously through a convex program with a throughput constraint. The algorithm is extended to handle discrete frequency levels by rounding up the results to a limited set of discrete frequencies. There are two problems with this approach. First, the assumption that frequency is proportional to dynamic voltage while performing VFS is unrealistic. Second, the handling of discrete frequency levels by rounding up the results of a continuous distribution can lead to a severe loss of energy savings, as we will show.

3. PRELIMINARIES

We now present our target platform, our data flow model for streaming applications, its timing properties, and the power model.

3.1 Target Platform

We consider heterogeneous multiprocessors, where both general purpose and application-specific cores have local memories and are connected along with peripherals (such as I/O ports) via a Network-On-Chip.

On our modem, the platform has 3 types of processors: Embedded Vector Processors (EVPs) [17], Decoders and ARMs, each of which may have multiple instances,

Formally, the multiprocessor $\prod(\Pi, F, p)$ has a processor set $\Pi = \{\pi_0, \pi_1, ...\pi_m\}$, where π_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 the power dissipation associated with a frequency level for each processor $p : \Pi \times F \rightarrow \mathbb{R}_0^+$. Two variations of the architecture are considered. In the first, frequencies and voltages are managed by a local a VF-switch per processor. In the second, a single global VF-switch simultaneously shifts the VF level for all processors.

3.2 Data Flow Model

We use SRDF [7] to model real-time streaming applications. An SRDF application graph $G = (V, E, d, t)$ is a directed graph . Nodes (actors) in V represent actual tasks and edges in E represent communication channels. Data is transported in discrete chunks, called tokens. An actor is enabled by the availability of one token on each of its incoming edges. An enabled actor can fire, consuming/producing from/to each of its input/output edges a single token. Actor firings are free from side effects. An iteration of the graph corresponds to one firing of every actor. Actors in a SRDF graph can be executed infinitely often. The valuation $d : E \rightarrow \mathbb{N}_0$ represents the initial token distribution on an edge. Thus, $d(i, j)$ represents the number of initial tokens (or delay) on edge (i, j). A SRDF is timed when the execution time of each actor is given by valuation $t : V \rightarrow \mathbb{R}^+$.

An edge (i, j) together with delay $d(i, j)$ impose a precedence constraint [16], which can be formalized as follows:

$$s(j, k) \geq e(i, k - d(i, j)), \forall k \geq d(i, j) \tag{1}$$

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.

For a timed SRDF graph G, the throughput of the system is given by the inverse of the Maximum Cycle Mean (MCM) [16] of G, defined as

$$\max_{c \in C(G)} \left(\sum_{i \in V(c)} t(i) \Big/ \sum_{e \in E(c)} d(e) \right) \tag{2}$$

where $C(G)$ is the set of cycles in G, $V(c)$ and $E(c)$ are the sets of actor and edges traversed by cycle c.

To handle infinite schedules, we restrict ourselves to static periodic schedules, specified by a triple:

$$SPS = \{\pi(i), s(i), \mu_d\}, \forall i \in V \tag{3}$$

where $\pi(i)$ and $s(i)$ represent the processor where actor i is mapped and the start time of the first firing of actor i, respectively, and μ_d stands for the SPS period. For SPS, strict periodicity is enforced by requiring that

$$s(i, k) = s(i, 0) + \mu_d \times k, \forall i \in V. \tag{4}$$

An SPS period μ_d for a SRDF is admissible only when $\mu_d \geq MCM$ [9]. Recall Equation 1, for SPS, the precedence constraint imposed by edge (i, j) becomes:

$$s(j, k) - e(i, k) \geq -\mu_d \times d(i, j), \forall k \geq d(i, j) \tag{5}$$

Given a SRDF graph, the precedence constraints introduced by a static ordering can be represented as additional edges E_S in the graph, and $E_S \bigcup E$ essentially implies the static ordering of task firings 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 π_1 and π_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.

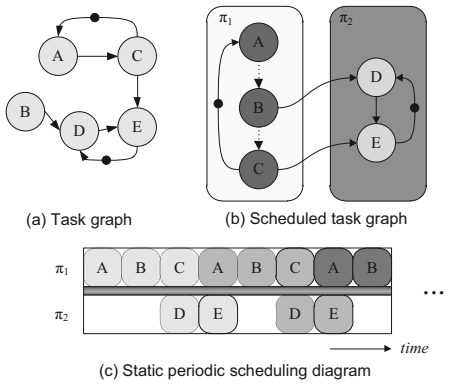

Figure 2: SPS for SRDF

3.3 Power And Delay Models

In this paper, we only consider dynamic power, for which the standard model [5] is used:

$$P_{dynamic} = \alpha C v_{dd}^2 f \tag{6}$$

where α models the circuit switching activity, C is the switched capacitance and v_{dd} and f are the voltage and frequency levels, respectively. The voltage and frequency pair (v_{dd}, f) determines the execution mode of a processor. When performing VFS, v_{dd} scales with f [5], which is given by:

$$f = K \cdot \frac{(v_{dd} - v_{th})^{\delta}}{v_{dd}} \tag{7}$$

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

4. MOTIVATIONAL EXAMPLE

We demonstrate the influence of infinite overlapped scheduling and cross-iteration data dependencies on VFS by means of an example, where we estimate the energy consumption of an application in three cases. In the first, the processors always run at the highest frequency when active, and are switched off when not active (consume no energy). In the second, we adapt an existing deadline-constrained VFS policy [4] to our example. In the third, we directly address the infinite SPS, and consider cross-iteration precedence constraints when scaling voltage and frequency. For clarity, we assume throughput-constrained VFS with discrete frequency levels.

The hardware platform has three processors π_1, π_2 and π_3, with their relevant parameters detailed in Table 1. Two

voltage-frequency levels are available to each. Power dissipations are made integer for simplicity sake. The virtual processor $\pi_{virtual}$ is used to map external non-scalable source tasks, and only the higher frequency is available to it. The processors need to execute tasks (S, A, B and C) with precedence relations shown in Fig. 3(a). Task S is the external source. All others are processing tasks.

All tasks initially execute in the high power/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 $E_{total} = 2 + 3.2 + 0.6 = 5.8\mu J$ (we assume processors consume no energy when inactive).

Table 1: Example platform description

Processor	Voltage(V)	Frequency(MHZ)	Power(mW)
π_1	1.1	312	200
	0.9	156	60
π_2	1.1	312	160
	0.9	156	50
π_3	1.1	312	20
	0.9	156	7
$\pi_{virtual}$	1.1	312	0

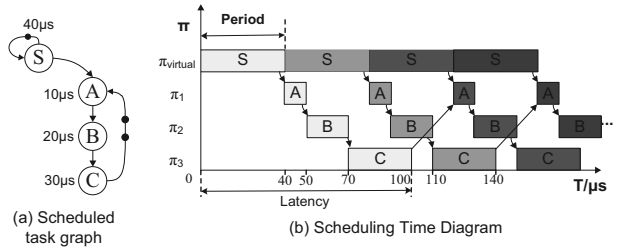

Figure 3: Application Example

For comparison purposes, we adapt an existing deadline-constrained VFS policy [4] to our example. As mentioned, schedules for real-time streaming applications have infinite lengths and are often overlapped across successive iterations. Hence, traditional deadline-constrained VFS policies [4] cannot be directly applied because they are restricted to one iteration. To reduce the size of the VFS problem, from infinite to finite, we split the infinite schedules into minimum static periodic regions (blocked schedules) and then apply deadline-constrained VFS policies to the 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 [4] 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 retimed. Retiming [22] redistributes the original tokens in a SRDF to improve blocked schedules. From this we obtain a blocked schedule, that can execute in a periodic fashion (Fig. 4(a)). Since precedence relations for a blocked schedule can be rearranged within each block, a *Direct Acylic 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 VFS with a deadline constraint of $40\mu s$. In the result, task A and task C extend execution times to $20\mu s$ and $40\mu s$, respectively. Task B is not scaled. The total energy consumption for one period is $E'_{total} = 1.2+3.2+0.54 = 4.94\mu J$.

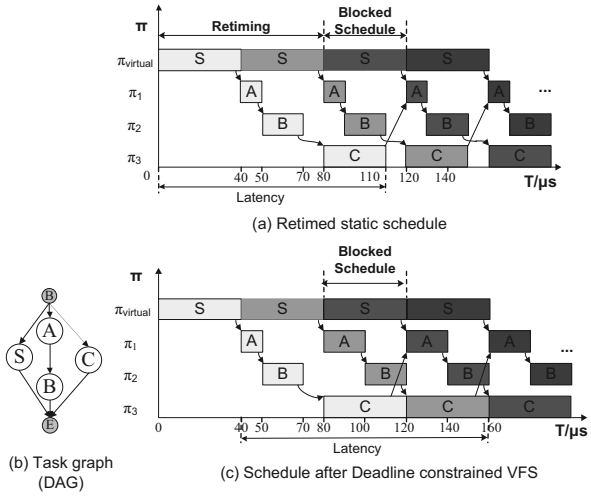

Figure 4: Transformed deadline-constrained VFS.

However, by directly addressing the infinite SPS and complying to cross-iteration precedence constraints stated in Equation 5, we can get the optimal scaling results meeting the period constraint of $40\mu s$. As is shown in Fig. 5, after scaling, task A and task B have the new execution times as $20\mu s$ and $30\mu s$. The total dissipated energy per period is $E''_{total} = 1.2 + 2.6 + 0.6 = 4.4\mu J$.

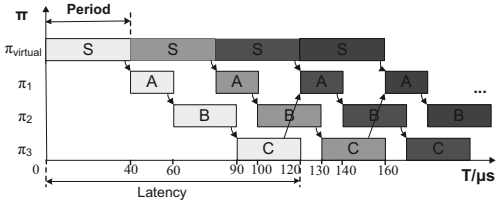

Figure 5: Throughput-constrained VFS

This example shows that, by directly addressing the throughput-constrained VFS problem, we obtain extra energy savings of $\frac{4.94\mu J - 4.4\mu J}{5.8\mu J} = 9.3\%$ 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[1] 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 after unfolding. Hence, we need algorithms that directly address throughput-constrained VFS.

5. PROBLEM FORMULATION

Our VFS techniques assume that task-to-processor mapping and static ordering of tasks per processor have already

[1]Unfolding [12] schedules N iterations of a SRDF graph together, which often leads to improved blocked schedules.

been performed by a scheduling flow such as the one proposed in [9].

As already discussed, streaming applications are different from deadline-constrained applications in 3 aspects: infinitely overlapped schedule, cross iteration data dependencies and both throughput and latency constraints. We use static periodic schedules to reduce the problem size from infinite to finite. Furthermore, cross-iteration data dependencies are naturally represented by edges with delays in SRDF, as demonstrated in Equation 5. To cope with deadline we explicitly express deadline (latency) constraints in terms of the throughput constraint in a SRDF model [9] by performing a graph transformation described in [9].

We now define the Throughput-Constrained VFS (TC-VFS) problem.

PROBLEM 5.1. ***Throughput-constrained VFS:*** *A SRDF $G(V, E, d, t)$ with throughput requirement μ_d^{-1} and a heterogeneous multiprocessor $\prod(\Pi, F, p)$ are given. The multiprocessor 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 given for the base frequency $f_b \in F$. For each task $i \in V$, the clock cycles count 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 of the processors, $\Pi_S \subset \Pi$ (used to map the external sources), are assumed to consume zero power. Task mapping is represented as $\pi : V \to \Pi$ and static ordering represented as $\sigma : \Pi \to \alpha^n$, where $\alpha^n = [i_1, i_2, ..., i_n]$ is the set of actor firing sequences of a processor, and $i_1, i_2, ..., i_n \in V$. Both task mapping and ordering are given a priori. The problem is finding a static periodic schedule with scaled task execution times using VFS, such that total energy consumption is minimized under the throughput requirement. The solution to each VFS variant is a static periodic schedule of voltage-frequency points for each VF-switch in the system and a static periodic schedule $\{\pi(i), s(i), \mu_d\}$ with scaled task execution times with period μ_d.*

In this paper, we study three variants of TC-VFS:

Table 2: Problem Variants

Problem	Frequency	VF-switch
TC-CLVFS	Continuous	Local
TC-DLVFS	Discrete	Local
TC-DGVFS	Discrete	Global

We solve TC-CLVFS as a convex program, TC-DLVFS as a MILP. For TC-DGVFS, we propose a 3-stage heuristic.

6. CONTINUOUS LOCAL VFS

throughput-constrained Continuous VFS with local VF-switches was proposed in [10], where it is formulated as a convex program. However, it assumes that frequency is proportional to dynamic voltage, which is unrealistic. Instead, we use the frequency and dynamic voltage relation 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 in polynomial time complexity [11].

PROBLEM 6.1. ***Throughput-Constrained Continuous Local VFS (TC-CLVFS)*** *is a TC-VFS problem where the frequency set includes all frequencies within a fixed range and each processor has its own VF-switch.*

The following TC-CLVFS formulation minimizes the total dynamic energy consumption by finding the optimal executing frequency for each task. $\pi(i)$ is processor mapped to task i, while $f(i)$ and $v(i)$ are 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 [14]. 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 :

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

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

$$s(i) \geq 0$$

$$v_{ddmin}(\pi(i)) \leq v_{dd}(i) \leq v_{ddmax}(\pi(i))$$

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

Notice that in the above formulation, only task mapping and ordering (represented as $E \bigcup E_S$) 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.

7. 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 [10], 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 can optimally solve this variant of TC-VFS by formulating a MILP program.

PROBLEM 7.1. ***Throughput-Constrained Discrete Local VFS (TC-DLVFS)*** *is a TC-VFS problem where the frequency set is finite and discrete, and processors scale operating frequency independently.*

THEOREM 7.1. *Problem 7.1 is NP-hard.*

Proof Sketch The Deadline-Constrained Discrete Local VFS (DC-DLVFS) problem is NP-hard [4]. By showing that DC-DLVFS can be reduced to a subset of TC-DLVFS in polynomial time, NP-hardness is proved.

In the following, we give a MILP formulation for TC-DLVFS. The goal is to optimally assign the cycles of each task to each frequency level, which consequently minimizes the total energy consumption. Energy consumption is given by 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 that 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 clock cycle count of each task is constant. For non-scalable tasks, execution can only be done on the base frequency f_b. Since all the constraints are linear, the program is a MILP. MILPs are in general NP-complete [15] and require non-polynomial time to solve. However, by dropping the constraint that variable $nc(i, f)$ be integers, we get a tight approximation algorithm as a linear program, which can be optimally solved in polynomial time.

TC-DLVFS algorithm

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

$$subject\ to:$$

$$\forall (i,j) \in E \bigcup E_S, \quad s(j) - s(i) \geq -d(i,j) \times \mu_d + \sum_{f \in F} \frac{nc(i,f)}{f}$$

$$\forall i \in V, \forall f \in F, \quad nc(i,f) \in \mathbb{N}$$

$$\forall i \in V, \quad s(i) \geq 0$$

$$\forall i \in V \setminus V_S, \quad \sum_{f \in F} nc(i,f) = nc(i)$$

$$\forall i \in V_S, \quad nc(i, f_b) = nc(i)$$

8. DISCRETE GLOBAL VFS

In the previous two sections, we solved TC-VFS problem variants that assume the platform has a dedicated VF-switch per processor. In many practical multiprocessors, however, a single global VF-switch is available to the whole multiprocessor due to cost and design simplification issues. In this section, we focus on that variant of the problem.

PROBLEM 8.1. ***Throughput-Constrained Discrete Global VFS (TC-DGVFS) :*** *A TC-DGVFS problem is a TC-VFS problem where the frequency set consists of a limited number of discrete values and, at any point in time, active tasks must run at the same voltage level.*

THEOREM 8.1. *Problem 8.1 is maximal open*[2].

Proof Sketch The single machine discrete time-cost scheduling problem [6] is maximal open and can be reduced to a subset of TC-DGVFS, with a single processor. Hence Theorem 8.1 holds.

With a global VF-switch, tasks running at the same time on different processors must switch frequency simultaneously. The crucial observation here is that we are scaling parallel blocks at a coarser granularity. By a Parallel Block (PB), we mean the shortest time interval of a schedule within which the set of running tasks remains constant. Consequently, the search for a VFS solution can be done in three stages: (1) find a blocked schedule with the shortest schedule length; (2) identify PBs in the schedule; (3) perform VFS for the schedule on PBs. The first stage finds a blocked schedule with the maximum amount of static slack. This slack will be used to stretch task executions. In the third stage, PBs extend their execution times under the original period constraint to save energy. Since in each block a different number of processors is active, we assign more slack to blocks where power dissipation is higher.

Lets illustrate this by recalling the example in Section 4. Our global discrete VFS algorithm works as follows: first,

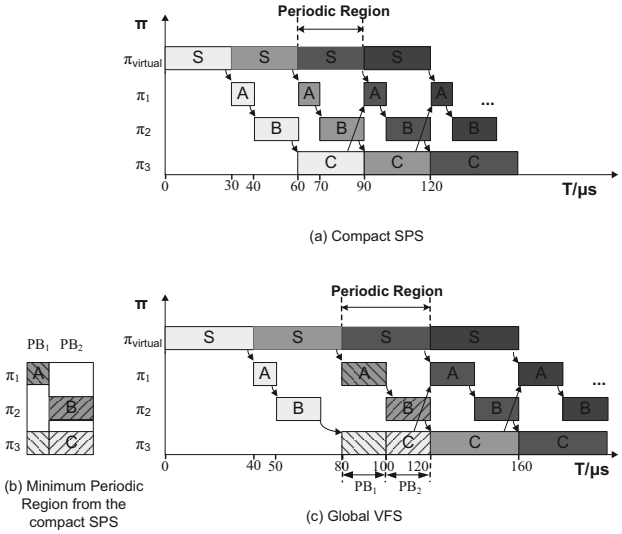

Figure 6: Throughput-constrained VFS

the original SPS schedule as shown in Fig. 3 is compacted using a source with a shorter period. The compact schedule (Fig. 6(a)) is then split into minimum periodic regions. Each region (Fig. 6(b)) is further divided into PBs, which are then scaled as if each one executes on a single processor with a deadline constraint ($40\mu s$) converted from the original throughput requirement. Since the source task is non-scalable, PBs are scaled such 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 $E''' = 1.2 + 3.2 + 0.54 = 4.94\mu J$.
Our heuristic for TC-DGVFS is given in pseudo-code below. First, it finds a blocked schedule of parallel blocks with maximum global static slack at the end, by finding an SPS for the smallest period the graph (without the external source) can sustain. Step 1 tries to shrink the source such that the resulting graph achieves the lower bound MCM of the SRDF G' consisting only of processing tasks. Step 2 computes an SPS for the resulting graph . Step 3 identifies parallel blocks on the compact SPS, and Step 4 determines timing and power characteristics per block. Step 5 finds a VF level for each parallel block in the blocked schedule. This is accomplished by a MILP program (also shown below).

In the formulation above, $nc(i, f)$ is the number of clock cycles parallel block i spent on frequency f, while $p_{pb}(i, f)$ is its power dissipation at frequency f. Stage 5. is identical to performing VFS for chained applications on a single processor. Since sources are non-scalable, the set of parallel blocks with the same spread source ς, PB_ς, must sum up to

[2]Maximal open [3] problems have unknown complexity but all harder cases are NP-hard.

Pseudo-code: TC-DGVFS algorithm

- -Find 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 $i \in V_S$
    $t(i) = \frac{t(i)}{SD} (SD \geq 1)$
  Endfor
  $SD{+}{+}$
Endwhile

2: Compute a new SPS for the updated graph G

- -Identify 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 dissipations of all tasks $i \in j$
    $p_{pb}(j, f) = \sum_{i \in j} p(i, f)$
  Endfor
Endfor

- -VFS for identified parallel blocks from a BS

5: VFS for PB, formulated as MILP program

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

$$minimize \sum_{i \in PB} \sum_{f \in F} p_{pb}(i, f) \times \frac{nc(i, f)}{f}$$

subject to :

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

$$\forall \varsigma \in V_S, \forall i \in PB_\varsigma, \quad \sum_{f \in F} \frac{nc(i, f)}{f} = \frac{nc(\varsigma)}{f_b}$$

$$\forall i \in PB, \forall f \in F, \quad nc(i, f) \in \mathbb{N}$$

the original execution time of source ς after scaling.

9. EXPERIMENTS AND RESULTS

We conducted experiments using four radio applications: WLAN, TDSCDMA, AM Radio and ChannelEq. The first two are taken from [9], and both include throughput and latency requirements (modeled in the graphs). The radios are scheduled onto the modem platform. For WLAN and TDSCDMA, only 45% of the processor cycles are available to each, as these applications share processors using a TDM scheduler. The effect of the TDM scheduler on the response of each task is modeled in data flow with the method proposed in [9]. The technology dependent parameters of the processors correspond to a 45 nm process. Energy consumption comparisons assume that if VFS is not applied, processors switch off when inactive with zero consumption and consume at the highest frequency when active. We used two solvers. For TC-CLVFS (a convex program), we use LINGO [2]. For MILP we used GLPK [1].

We evaluate our VFS techniques for all four radios. The results are shown in Table 3 and Fig. 7. We run all 3 variants. The discrete algorithms use 5 frequency levels. The set of frequency levels is a geometric series with a common ratio of 2.

Table 3: Results for radio applications

Application	Energy Reduction (%)		
	TC-CLVFS	TC-DLVFS	TC-DGVFS
WLAN	29	28	16
TDSCDMA	37	36	20
AM Radio	10	6	1
ChannelEq	15	10	4

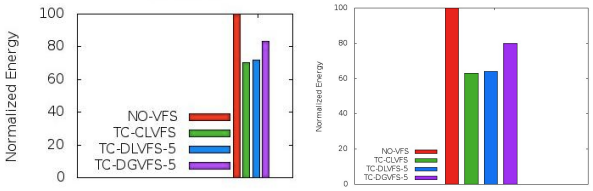

Figure 7: Comparison of TC-VFS algorithms

For WLAN and TDSCDMA, the results show that our algorithms significantly reduce energy consumptions. For WLAN, the TC-CLVFS algorithm saves 29% energy, while the TC-DLVFS and TC-DGVFS algorithms save 28% and 16%, respectively. The energy reductions for these applications are due to large amounts of static slack. The results for AM Radio and ChannelEq show that, for applications with tight schedules (i.e. few static slack), energy saving differences among algorithms can be significant. For AM Radio, the TC-CLVFS algorithm saves 10% energy, while the TC-DLVFS and TC-DGVFS algorithms save 6% and 1%, respectively. When there is few static slack, 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 to the highest frequency level, which saves energy poorly. Moreover, the performance of our discrete global algorithm further depends on how much global static slack exists in a compact schedule (recall Section 8). Since AM Radio works mainly in a chained fashion under a tight schedule, the discrete global algorithm cannot efficiently use static slack compared to the discrete local algorithm. Thus, local VF-switches have clear advantages.

The second set of experiments (Table 4) compares our TC-DLVFS algorithm versus the technique proposed in [10], which uses discrete frequency levels by rounding up the results of a continuous VFS algorithm, i.e., the frequency per task is conservatively rounded up to its closest available frequency.

Table 4: Comparison with rounding up

Application	Energy Consumption ($10^{-6}J$)		
	NO-VFS	Round-up	TC-DLVFS
WLAN	1.15	1.14	0.83
TDSCDMA	24.81	16.51	15.91

The results show that the rounding heuristic can cause severe loss of savings. For WLAN, it almost saves no energy (1%) while our optimal TC-DLVFS algorithm reduces it by 28%. This is because many frequency levels obtained by the TC-CLVFS algorithm are between the highest two frequency levels, which the heuristic rounds to the highest frequency. For TDSCDMA, the heuristic and our TC-DLVFS

algorithm save energy consumption by 33% and 36%, respectively. This supports our claim that directly addressing discrete frequencies by our TC-DLVFS algorithm can in some cases improve results substantially.

Finally, we evaluate the effects of the number of voltage levels on our discrete local and global VFS algorithms. We use 1 to 5 frequency levels, where 1 level means no VFS and n levels means that the highest n frequency levels in the system are used. Fig. 8 shows the results for both WLAN and TDSCDMA. We can see that energy consumption is reduced dramatically from 1 to 3 frequency levels, and continues to decrease slightly after that. This is due to the convex relation between energy consumption and frequency [21]. Energy consumption increases dramatically fast when frequency is high, thus the majority of energy savings are done through the initial stages of scaling down frequency.

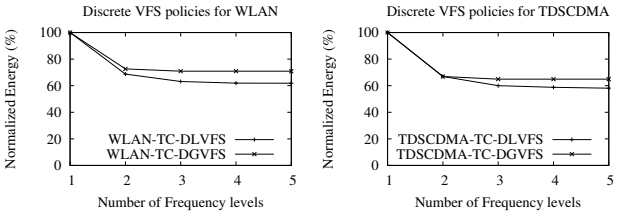

Figure 8: Impact of #discrete frequency levels

10. CONCLUSION

Throughput-constrained VFS problems are different from deadline-constrained VFS in that they have to deal with infinite schedules, cross-iteration dependencies and both deadline and throughput constraints. We studied throughput-constrained VFS variants, evaluating the impacts of continuous or discrete frequency levels, and of local VF-switches per processor against a global VF-switch. We demonstrated that convex programming and mixed integer linear programming can be used to solve these problems. We studied the complexity of the problem variants, concluding that the discrete local variant is NP-hard, and that the discrete global variant has maximal open complexity.

The experiments we conducted on transceivers running on our reference modem architecture showed that our proposed VFS algorithms can effectively reduce energy consumptions, up to 28% for discrete local VFS, and 16% for discrete global VFS when compared to a on/off energy saving policy. We have also established that there is in some cases a considerable advantage in terms of energy savings, when using local VF-switches instead of a global VF-switch.

As future work, we intend to focus on addressing VFS for data flow graphs with conditional behavior, such as the Mode-Controlled graphs proposed in [9], and on optimizing VFS for self-timed task synchronization, instead of relying on fully static schedules.

11. REFERENCES

[1] Glpk. `http://www.gnu.org/software/glpk/`.

[2] Lingo. `http://www.lindo.com/index.php`.

[3] M. Aloulou, M. Y. Kovalyov, and M. Portmann. Evaluating flexible solutions in single machine scheduling via objective function maximization: the study of computational complexity. *RAIRO - Operations Research*, 2007.

[4] A. Andrei et al. Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems. In *DATE*. IEEE, 2004.

[5] A. P. Chandrakasan et al. *Low Power Digital CMOS Design*. Kluwer, 1995.

[6] Y. He, Q. Wei, and T. C. Cheng. Single-machine scheduling with trade-off between number of tardy jobs and compression cost. *Journal of Scheduling*, October 2007.

[7] E. Lee and D. Messerschmitt. Synchronous data flow: Describing signal processing algorithm for parallel computation. In *COMPCON*, 1987.

[8] H. Liu, Z. Shao, M. Wang, and P. Chen. Overhead-aware system-level joint energy and performance optimization for streaming applications on multiprocessor systems-on-chip. ECRTS '08.

[9] O. Moreira. *Temporal Analsis and Scheduling of Hard Real-Time Radios on a Multi-processor*. PhD thesis, Eindhoven University, Netherlands, 2012.

[10] A. Nelson et al. Power minimisation for real-time dataflow applications. In *EUROMICRO DSD*, 2011.

[11] Y. Nesterov et al. *Interior-Point Polynomial Algorithms in Convex Programming*. Studies in Applied and Numerical Mathematics. Society for Industrial and Applied Mathematics, 1987.

[12] K. Parhi and D. Messerschmitt. Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding. *IEEE Trans. Comput.*, Feb. 1991.

[13] R. Rockafellar. *Convex analysis*. Princeton Mathematical Series. Princeton University Press, 1997.

[14] K. Seth et al. Fast: Frequency-aware static timing analysis. *ACM TECS*, Feb. 2006.

[15] S. Sinha. *Mathematical Programming: Theory and Methods*. Elsevier Science Ltd, 2006.

[16] S. Sriram and S. Bhattacharyya. *Embedded multiprocessors*. Marcel Dekker, 2000.

[17] K. van Berkel et al. Vector processing as an enabler for software-defined radio in handheld devices. *EURASIP J. Appl. Signal Process.*, January 2005.

[18] Y. Wang et al. Overhead-aware energy optimization for real-time streaming applications on multiprocessor system-on-chip. *ACM TODAES*, 2011.

[19] R. Xu, R. Melhem, and D. Mosse. Energy-aware scheduling for streaming applications on chip multiprocessors. RTSS, 2007.

[20] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Symp. on Foundations of Computer Science*. IEEE, 1995.

[21] Y. Zhang, X. S. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *Proc. DAC*. ACM, 2002.

[22] V. Zivojnovic and R. Schoenen. On retiming of multirate dsp algorithms. In *Proc. IEEE Conference on Acoustics, Speech and Signal Processing*, 1996.