Power Minimisation for Real-time Dataflow Applications

Andrew Nelson[1], Orlando Moreira[2], Anca Molnos[1], Sander Stuijk[3], Ba Thang Nguyen[1] and Kees Goossens[3]
[1]Delft University of Technology, Delft, The Netherlands
[2]ST Ericsson, Eindhoven, The Netherlands
[3]Eindhoven University of Technology, Eindhoven, The Netherlands
a.t.nelson@tudelft.nl, orlando.moreira@stericsson.com, a.m.molnos@tudelft.nl, s.stuijk@tue.nl, k.g.w.goossens@tue.nl

***Abstract*—Energy efficient execution of applications is important for many reasons, e.g. time between battery charges, device temperature. Voltage and Frequency Scaling (VFS) enables applications to be run at lower frequencies on hardware resources thereby consuming less power. Real-time applications have deadlines that must be met otherwise their output is devalued. Dataflow modelling of real-time applications enables off-line verification of the application's temporal requirements. In this paper we describe a method to reduce the combined static and dynamic energy consumption using a Dynamic VFS (DVFS) technique for dataflow modelled real-time applications that may be mapped onto multiple hardware resources. We achieve this by using an application's static slack in order to perform DVFS while still satisfying the application's temporal requirements. We show that by formulating a dataflow modelled application and its mapping as a convex optimisation problem, with energy consumption as the objective function, the problem can be solved with a generic convex optimisation solver, producing an energy-optimal constant frequency per application task. Our method allows task frequencies to be constrained such that, e.g. one frequency per application or per processor may be achieved.**

I. Introduction

In general, reducing the energy consumption of a system is desirable. Voltage and Frequency Scaling (VFS) is a commonly used method to reduce the energy consumption of electronic systems [1]–[5]. Lowering the voltage decreases the energy consumption but also lowers the frequency at which the system can run. Lowering the frequency of processing cores running real-time applications must therefore be done in the context of the applications' temporal requirements.

Modelling a real-time application as a Homogeneous Synchronous Dataflow (HSDF) graph [6], as illustrated in Figure 1, enables the off-line analytical verification of the application's throughput requirements. Using the Worst Case Execution Times (WCET) of the application's set of tasks, it is subsequently possible to calculate the minimum difference between the application's throughput, at a given frequency, and its throughput requirement. For a functional real-time application that meets its throughput constraints, we refer to this constant difference as *static slack*. It is the application's static slack that we use to perform VFS, or Dynamic VFS (DVFS) in the case that multiple tasks with different frequencies are mapped onto the same processor.

Our main contribution is a method to derive energy optimal task frequencies for Dataflow modelled applications using the *Disciplined Convex Programming* technique [7], for which problems are proven to be solvable in polynomial time. As an objective function we can either fit (or approximate) power consumption to frequency data that is either collected through experimentation, or derived from a model, as long as the data set is (approximately) convex [8], [9]. The convex program's optimisation constraints are derived from the real-time application's dataflow model. We contribute a dataflow modelled task execution scheme that enables task scheduling and communication timings to be taken into account.

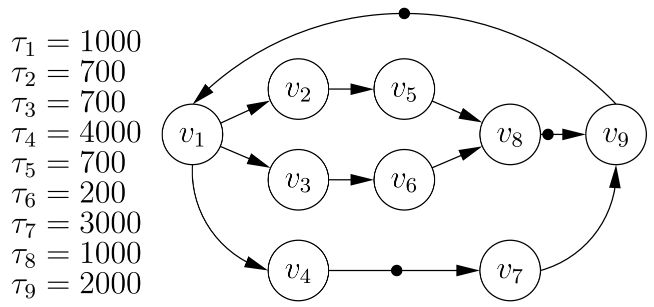

Figure 1. Example HSDF graph with timings τ_v in cycles.

We use a generic disciplined convex program solver called CVX [10] to derive the optimal task frequencies, which takes only a matter of seconds for our test applications, on an off-the-shelf laptop. While still maintaining the real-time application's throughput requirements, we are able to derive frequencies for energy reduction for various scenarios, e.g. 1) one frequency per task, 2) one frequency per processor, 3) one frequency for all tasks. This enables a trade off in the number of static voltage and frequency domains and the need for DVFS capability against the energy savings.

Our technique's limitations are that it is limited to using static-slack for energy reduction, e.g. Slack produced due to variation in task execution time, also known as dynamic-slack, is not used. Our method is limited to Dataflow modelled applications that have tasks that are bound by a WCET. Our technique is limited to single and multiprocessor Systems-on-Chip (SoCs) with predictable computation and memory access times. We do not take into account chip temperature when calculating the task operating frequencies.

We evaluate the practicality of our technique by apply-

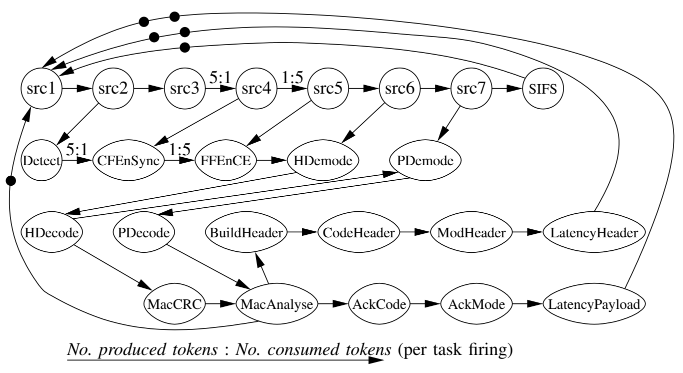

Figure 2. WLAN 802.11a SDF graph.

ing it to an actual tile-based, Multiprocessor SoC (MPSoC), hardware platform implementation, that uses per-tile DVFS hardware, as described in [11]. We apply our technique to both a synthetic real-time dataflow application example, as illustrated in Figure 1, and a WLAN 802.11a, real-time, dataflow application that can be found in [12] and illustrated in Figure 2. We show that our technique uses the available static-slack to enable energy conservation through operating frequency reduction.

The rest of this paper is structured as follows. In the next section we put our work into the context of previous work that has been carried out in the direction of energy reduction of embedded systems through DVFS. We follow this in Section IV with a detailed explanation of how to formulate the relevant properties of an HSDF structured application along with an energy consumption model for convex optimisation. In Section V we explain how hardware resource constraints may be taken into account using our technique. We present experimentation to verify our claims in Section VI and finish off with concluding statements in Section VII.

II. Related Work

There is already quite a large body of work available that demonstrates how DVFS techniques may be applied to reduce energy consumption. While it would not be possible to reference all published work that use such techniques, due to space restrictions, we provide a selection of relevant related work to the topics encountered in this paper. We start by relating other techniques that use formal optimisation methods to achieve energy reduction through the use of DVFS. We further relate work that incorporates a DVFS technique in a Real-Time Operating System (RTOS). We finish this section by relating work that uses timed Dataflow analysis to provide energy reduction through DVFS for real-time applications.

Methods for power optimisation in SOC-based Dataflow systems are described in [5]. The work applies linear programming methods to derive optimal frequencies of hardware components to reduce energy consumption. The work in [5] does not put forward a technique that can be used at the application level. This can be seen as a simpler problem than the one we tackle due to the inherently parallel nature of hardware.

The work in [1] has a much larger scope than ours, tackling task allocation, scheduling and voltage scaling. Focussing on the VFS part of their work, the frequency levels are obtained using an Integer Linear Programming (ILP) method, through traditional branch and bound. While the complexity of their technique is not stated in their work, ILP methods may be NP-Hard in complexity. Our method uses Disciplined Convex Programming which is proven to have a polynomial complexity [7].

Leakage aware multiprocessor scheduling is described in [2]. DVFS is applied to applications modelled as directed acyclic graphs with the nodes mapped onto a multiprocessor. A technique called schedule and stretch is used to apply DVFS. This is achieved by first scheduling the tasks, calculating the amount of slack that remains in the schedule before the deadline, then frequency scaling the MPSoC until the deadline is just met thus saving energy. This work in comparison to ours applies a uniform frequency scaling to the entire MPSoC. In our work each core has an independent DVFS ability enabling potentially larger energy savings. Our method is also not restricted to acyclic application graphs.

In [3] a Real-Time Operating System (RTOS) is presented based on [13] which uses a "cooperative voltage scaling" method where the RTOS and the application collaborate to perform DVFS. Their method uses fixed size slice lengths, using the tasks WCET to calculate the frequency required to complete the task in its assigned number of slices. Their work is only applied to a single core whereas ours is multi-core. They calculate the frequency of each task within the context of its own deadline whereas we calculate the frequency in the context of the entire application and its mapping on multiple processors.

Task-level dataflow modelling has been investigated before for the purposes of using DVFS for real-time applications, in order to reduce energy consumption. In [14], [15] methods were proposed that used a combination of Hierarchical FSM modelling with SDF [6] modelling for the purposes of intra-task VFS. Their method relies on identifying the application's execution trace so that the tasks on that trace may be subjected to VFS. Our method works for dataflow models that may be converted to HSDF models, such as SDF and cyclo-static SDF [16] and uses inter-task VFS. We also show that our method works for multi-processor platforms.

III. Background

For our method we require a hardware platform that provides predictable computation and communication. For this purpose we use the platform template described in [11]. The platform template uses a tile based hardware architecture, with tiles connected using a predictable hardware interconnect, such as those described in [17]–[19]. The platform implementation that we use for experimentation follows this template using the Æthereal Network-on-Chip (NoC) [17] to connect a configurable number of Xilinx [20] MicroBlaze (μBlaze) processor based tiles, as illustrated in Figure 3. One of the μBlaze tiles acts as the host tile, configuring the NoC and monitoring status information received from the other tiles.

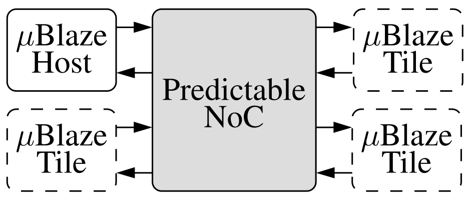

Figure 3. Hardware platform architecture.

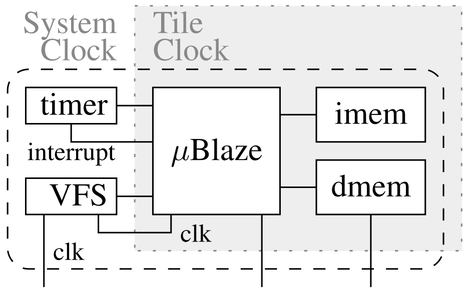

Figure 4. MicroBlaze (μBlaze) based processing tile.

The rest of the tiles consist of a μBlaze processing core connected to a programmable timer that sends an interrupt after its programmed time delay and a VFS unit, as described in [11], [21]. The DVFS module provides 17 equidistant power levels from a 0 MHz clock gate state up to the frequency provided by the system clock, as illustrated in Figure 4. As per the scheme described in [11] DMA's are used to transfer data between tiles, with at least one DMA per application to ensure application level predictability.

Predictability is also important on the processor. For our DVFS technique to be applicable it must be possible to guarantee the amount of processing time an application has in a fixed period. To facilitate this our platform uses the CompOSe [22] Real-time Operating System (RTOS) with Time Division Multiplexed (TDM) scheduling to allocate processing time to an application guaranteeing that the application is scheduled for a fixed amount of time in the TDM period. We achieve this as described in [11] using a hardware timer connected to each processor that is programmed to interrupt the processor after fixed intervals allowing running applications to be swapped out by the RTOS in a fixed length time slice, which we refer to as the OS time slice. Importantly, during the application's allotted time no interrupt can arrive apart from that associated with the TDM scheduler signalling the end of its scheduled time. The RTOS controls the voltage and frequency of the tile using the DVFS module. Only the task execution times are scaled by the RTOS, the tile runs at f_{max} at all other times.

The relatively slow transitioning between Voltage and Frequency levels has long been an obstacle for DVFS techniques. Work in [23]–[26] demonstrate techniques where this is no longer the case. Taking the work in [25], [26] as an example, they demonstrate a fine grained DVFS technique with switching times in nano seconds. The work in [26] presents measurements from a physical implementation of the technique demonstrating that the technique is not just theoretical. For the work in this paper we do not use these techniques, but instead use a coarser grained DVFS module per-tile. A fine grained DVFS technique would enable our technique to use even more of an application's static-slack due to a smaller frequency rounding error, from the frequencies produced from our convex optimisation technique.

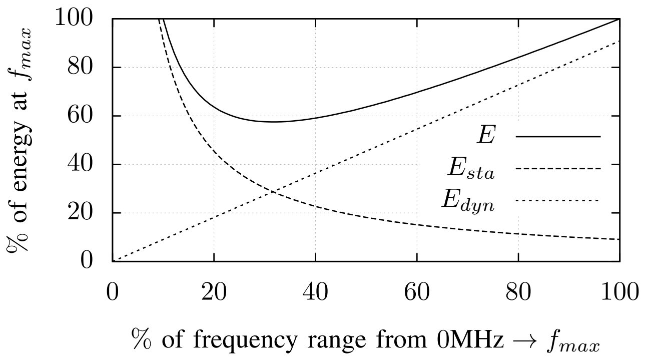

Figure 5. Normalised energy consumption against normalised frequency.

IV. Optimisation Formulation

Our energy reduction technique is aimed at Multiprocessor platforms that can provide applications with a guaranteed service rate regardless of influences external to the application, such as other applications, interrupts, etc. In order for our DVFS scheme to work there must be hardware support for DVFS per processor, meaning that the platform must be able to support multiple clock domains.

Our objective is to produce a frequency per task in order to reduce the amount of energy consumed while not violating the application's temporal requirements. To achieve this we use the generic convex optimisation [7] solver called CVX [10]. In order to solve the problem it must first be formulated as an objective function and a set of constraints. As an objective function for our technique it is possible to either fit (or approximate) power consumption to frequency data that is either collected through experimentation, or derived from a model, as long as the data set is (approximately) convex [8], [9]. As such our technique is not limited to any particular power model.

For the purpose of an illustrative example we approximate the convex energy per cycle curve produced from the energy model described in [2]. It takes into account both dynamic and static energy consumption using constants for a particular processor that has been processed in a $70nm$ technology. The energy consumed by a processor per cycle is the sum of the static E_{sta} and dynamic E_{dyn} energy consumption of the processor per cycle, i.e. $E = E_{sta} + E_{dyn}$. For our technique we can derive frequencies to minimise static or dynamic energy, or both. In general the combination of the two types of energy curve results in a convex curve similar to that found in [2]. As such we approximate energy per cycle as:

$$E = \frac{f_{max}^2}{f} + 10f \tag{1}$$

where f is the processor frequency and f_{max} is the maximum frequency at which the processor can run. Equation (1) produces the energy per cycle graph that is illustrated in Figure 5 (N.B. This energy model is used for demonstrative purposes throughout this paper. *Other models, or experimental data, that have a convex approximation may be used with our technique*). In order to calculate the energy consumption of an application it must be known for how many cycles the application will run at a particular frequency. This is a prerequisite for real-time applications.

Modelling the application as an HSDF graph [6] enables conservative analysis of the application to ensure that it meets its temporal requirements. An HSDF graph is a directed graph that can be used to represent an application, with *edges* representing fifo communication, and vertices, referred to as *actors*, representing application tasks, as illustrated in Figure 1. Data is communicated along the edges in discrete quantities called *tokens*. Tasks can start execution as soon as sufficient tokens are available on their incoming edges. This is referred to as an actor *firing*. Upon firing an actor consumes one token from each of its incoming edges and when it has finished produces one token on each of its outgoing edges.

For analytical purposes we represent an HSDF graph G using the tuple (V, E, τ, f, δ). Element V is the finite set of annotated vertices or actors. Element E is the finite set of annotated directed edges that connect the vertices. Edges are represented by the tuple $(i, j) \in E$ where $i \in V$ is the actor *producing* tokens on the edge and $j \in V$ is the actor *consuming* tokens from the edge. The WCET of an actor v is given by $\tau(v)$, with $\tau : V \rightarrow \mathbb{N}$. We annotate the graph with the WCET of tasks in terms of cycles. The processor frequency for the actor v is given by $f(v)$, with $f : V \rightarrow \mathbb{N}$. The WCET of an actor v in terms of seconds is therefore given by $\tau(v)/f(v)$. The initial token occupancy of an edge (i, j) is given by $\delta(i, j)$, with $\delta : E \rightarrow \mathbb{N}$.

The objective function can be formulated from (1) as follows:

$$minimise\left(\sum_{\forall v \in V} \tau(v) \cdot \left(\frac{f_{max}^2}{f(v)} + 10f(v)\right)\right) \tag{2}$$

where $f(v)$ are free variables in the optimisation. This is formatted for CVX accordingly:

```
minimise(sum(tau*(inv_pos(f)*fmax^2 + 10*f)));
```

with `tau` and `f` being horizontal and vertical vectors respectively. The `inv_pos` function is provided by CVX to find the inverse of positive numbers while indicating to the tool that the result is convex. The convex solver can derive the frequency or frequencies that produce the lowest energy consumption for the execution of the application. The derived frequencies must not cause the application to violate its temporal requirements. We achieve this by adding optimisation constraints that take into account the application's dataflow structure and temporal requirements.

The throughput of an HSDF graph is the inverse of its Maximum Cycle Mean (MCM) $\mu(G)$. The MCM of a graph G is calculated from its set of cycles $C(G)$, and is defined as:

$$\mu(G) = \max_{c \in C(G)} \mu(c)$$

where $\mu(c)$ is the cycle mean of the cycle c, defined as:

$$\mu(c) = \frac{\sum_{v \in V(c)} \tau(v)/f(v)}{\sum_{(i,j) \in E(c)} \delta(i, j)}$$

$V(c)$ and $E(c)$ are the sets of all actors and edges traversed by the cycle c respectively.

A necessary and sufficient condition for the feasibility of a periodic schedule is given in [27]. The constraint is applied per edge $\forall(i, j) \in E$ and is defined as:

$$s(i) - s(j) \leq T(G) \cdot \delta(i, j) - e(i) \tag{3}$$

where $s(v)$ is the start time of actor $v \in V$, $e(v)$ is the execution time of the actor v in seconds and $T(G)$ is the period of the schedule for application G. Equation (3) implicitly asserts the condition that there is no auto-concurrency of actor firings, meaning that multiple firings of an actor cannot occur at the same time. We extend (3), to take into account that the execution time of an actor is dependent on the frequency at which it is executed, as follows:

$$e(i) = \frac{\tau(i)}{f(i)}$$
$$s(i) - s(j) \leq T(G) \cdot \delta(i, j) - \frac{\tau(i)}{f(i)} \tag{4}$$

In [27] the WCET of tasks are constant as the frequency is constant ($f(i) = f_{max}$), making the constraint suitable for linear programming methods. Here however, the WCET in (4) is calculated using $\tau(i)/f(i)$, with $f(i)$ a free variable in the optimisation. Equation (4) is no longer a linear constraint making it unsuitable for linear programming methods, but it is suitable for the convex optimisation technique used by the CVX solver.

Equation (4) constrains the starting time of the consuming task $s(j)$ so that it can only start after the producing task $s(i)$ has finished execution. It is formatted for CVX as follows:

```
s(i)-s(j) <= T(G)*d(e) - tau(i)*inv_pos(f(i));
```

where `e` is the edge number of (i, j). It is applied for all edges in the application graph.

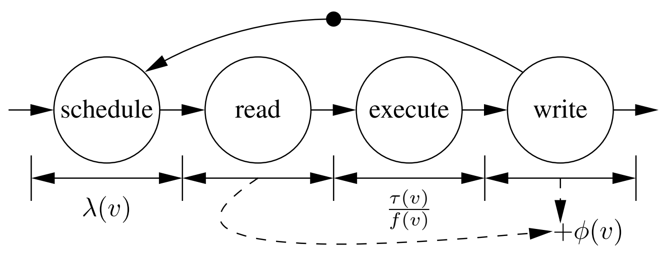

Figure 6. Dataflow representation of a task v firing on our platform, with associated timing notation.

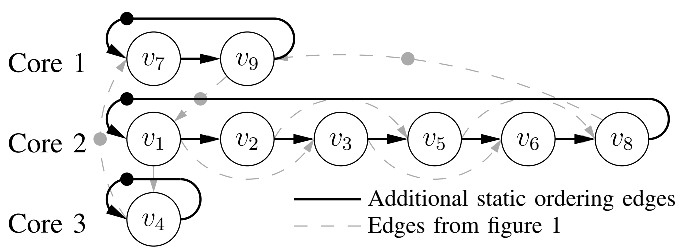

Figure 7. Additional static ordering edges for the HSDF graph from Figure 1 that is mapped onto three processors.

V. Formulation Applied to an MPSoC Platform

In the previous section we described our method to derive a frequency or frequencies to run an application at in order to reduce the energy consumption for the amount of work done. Our method, as described in the previous section, only takes into account the constraints of the application structure and is therefore rather idealised. In this section we show how our technique can be applied to an MPSoC platform based on the platform template from [11] taking into account relevant factors such as resource constraints and interprocessor communication time. As such, for interprocessor communication it must be possible to give a guaranteed temporal upper bound for any transaction, as is done by the Æthereal NoC [17]. We contribute a task execution scheme that is modelled as a HSDF graph, as illustrated in Figure 6. This dataflow task execution model is a one-to-one replacement of the application's HSDF graph's task nodes, enabling the worst case timings of task scheduling and communication to be taken into account while allowing the computation to be frequency scaled using our Disciplined Convex Programming technique.

Our RTOS supports multiple applications simultaneously that are scheduled using TDM. Once an application is scheduled its tasks are scheduled following a static order and only if it has fulfilled the HSDF firing rules, i.e. if their data dependencies have been met and there is space in their outgoing FIFOs. The pre-defined static order schedules can be constructed using well known SDF scheduling techniques, as found in [28]. The TDM task arbitration in combination with the static task execution ordering, and task execution times that have temporal upper bounds, enable a static upper bound to be derived for the scheduling time required for each task. The scheduling upper bound is given by $\lambda(v)$ with $v \in V$ and is incorporated into edge the constraint described in (4) as follows:

$$s(i) - s(j) \leq T(G) \cdot \delta(i,j) - \frac{\tau(i)}{f(i)} - \lambda(i) \qquad (5)$$

since in an HSDF structured application every task must be scheduled exactly once per graph iteration. The $\lambda(v)$ does not bound the interprocessor communication time of data between tasks mapped on different processors. We continue by explaining how this may be taken into account in our formulation.

After the task has been scheduled it reads its data to be processed from an external to a local memory, if it is not already located there. The task subsequently *executes* operating solely on the local data. Upon completion of the task execution the data that is produced is written from the local memory to a remote memory location, if required. We refer to the tasks *read* and *write* step as the task's communication.

For our DVFS technique we are only concerned with the execution part of the task execution scheme. This means that the task's communication always executes at maximum frequency. Due to the predictable nature of the interconnect, the task's communication has a predictable temporal upper bound. In a single iteration of the application's task graph each task's communication must happen once. This is incorporated in the constraint given in (5) as follows:

$$s(i) - s(j) \leq T(G) \cdot \delta(i,j) - \frac{\tau(i)}{f(i)} - \lambda(i) - \phi(i) \qquad (6)$$

with $\phi(v)$ being the WCET in seconds of the communication of task $v \in V$. Other timings associated with task execution that have temporal upper bounds, such as VFS transition time, can be taken into account in a similar fashion as to that which we have described for the task scheduling and communication.

The application execution is also constrained by the availability of processing resources. On a single processor only one task can execute at any time. For an application with a fixed task mapping onto processing tiles on an MPSoC, and also with static order execution of tasks on each tile, the additional timing constraints can be expressed using additional edges in the dataflow graph. Figure 7 illustrates how this may be achieved for a 3 core mapping, and static order scheduling of tasks, from the application graph illustrated in Figure 1. Similarly Figure 8 illustrates how extra edges may also be added to the more complex WLAN application SDF graph from Figure 2, which may be translated into an HSDF graph. The additional edges in the HSDF graphs are then expressed using the constraint formulation (6) for each of the new edges. By setting $T(G)$ equal to the inverse of throughput constraint of the application, i.e. setting $T(G)$ to be equal to the maximum valid MCM $\mu(G)$ of the application graph G, valid frequencies can be produced that meet the temporal constraints assuming the application is scheduled 100% of the time.

Multiple applications may run on our MPSoC platform, with the RTOS using TDM scheduling to ensure each application

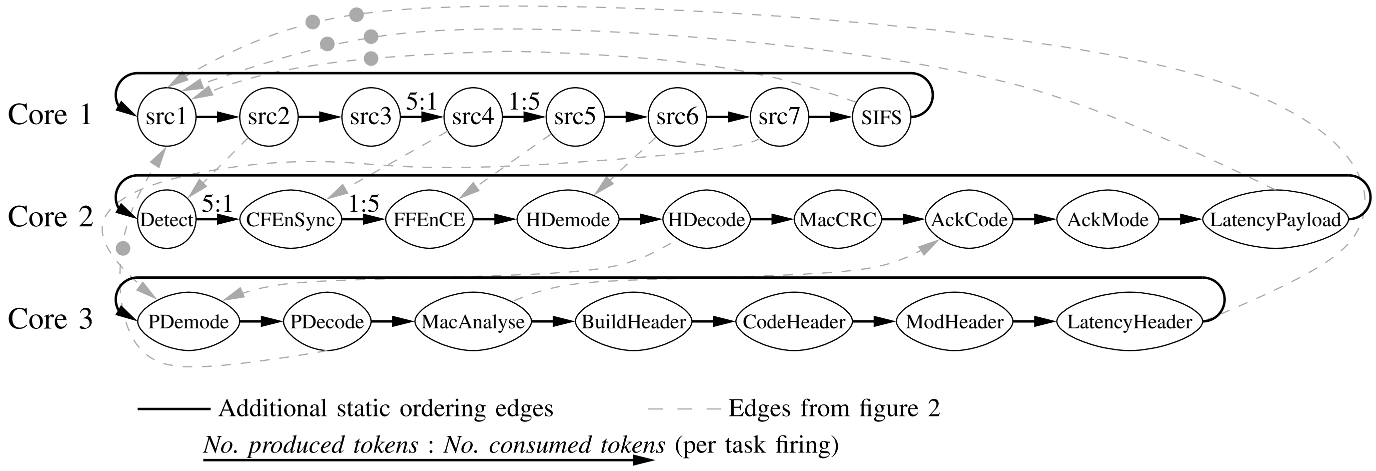

Figure 8. Additional static order edges for the WLAN SDF graph from Figure 2 that is mapped onto three processors.

receives a guaranteed fraction of the processor time in the fixed TDM period. This can be addressed in the optimisation constraints by scaling the maximum valid MCM of the graph linearly by the same fraction, meaning that higher frequencies are required to achieve the desired throughput if the application only has a fraction of the processor time compared to if it has all of it. For a TDM schedule with a period of N slots consisting of a fixed duration of R seconds for the RTOS to schedule the next application, and a fixed duration of L seconds for executing tasks. If application G is allotted M of the N slots then the scaled value of $T(G)$ is calculated as follows:

$$T(G) = \frac{\overline{\mu(G)} \times M \times L}{N \times (R + L)} \tag{7}$$

with $\overline{\mu(G)}$ being the maximum possible MCM of application graph G while still meeting the application's temporal requirements. For an application scheduled on multiple processors at a time this is method is valid for an application if the TDM tables have the same number of slots N, slot length L, are in phase and the application is assigned the same slots in each table. The optimisation produces frequencies in the continuous $\mathbb{R}$ domain while the processor can only handle a limited set of discrete frequency levels. To ensure that the throughput requirement is not violated we assign the closest higher frequency level from the reduced set, e.g. for a platform with Z frequency levels including 0 the discrete frequency $f(v)$ of a task $v \in V$ is calculated as follows:

$$f(v) = \left\lceil \frac{f_{\mathbb{R}}(v) \times (Z-1)}{fmax} \right\rceil \times \frac{fmax}{Z-1} \tag{8}$$

with $f_{\mathbb{R}}(v)$ being the timing in the continuous $\mathbb{R}$ domain derived using CVX.

We do not address buffer sizing as part of the optimisation problem. This is due to the fact that we derive task frequencies for an application for a given mapping and static order scheduling. The required buffer sizes are dictated by this input and can be calculated using the buffer sizing technique described in [27].

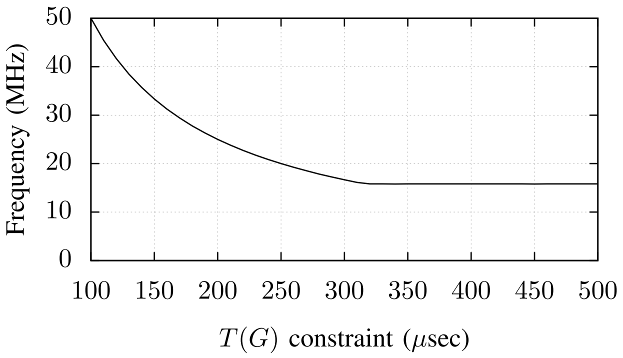

Figure 9. Application frequency by varying the throughput constraint.

VI. Experimentation

In the previous sections we describe how to formulate real-time dataflow applications as convex optimisation problems for a given task mapping on a MPSoC and static ordering of tasks on each processor. In this section we apply the theory described in the previous sections using experimentation, starting with the theory presented in Section IV. We follow this by showing how taking into account resource constraints affects the amount of energy reduction in comparison, i.e. experimentation to see how the theory in Section V affects the energy reduction when compared to the more ideal situation in Section IV. We subsequently test our methods by applying them to the synthetic real-time dataflow application illustrated in Figure 1 mapped onto an implemented MPSoC that has 3 μBlaze processing tiles with a $f_{max} = 50$ MHz, as illustrated in Figure 3. For the same platform we also evaluate our technique when used with a Wireless LAN 802.11a real-time dataflow application, as found in [12] and illustrated in Figure 2.

Using the HSDF graph of a synthetic application illustrated

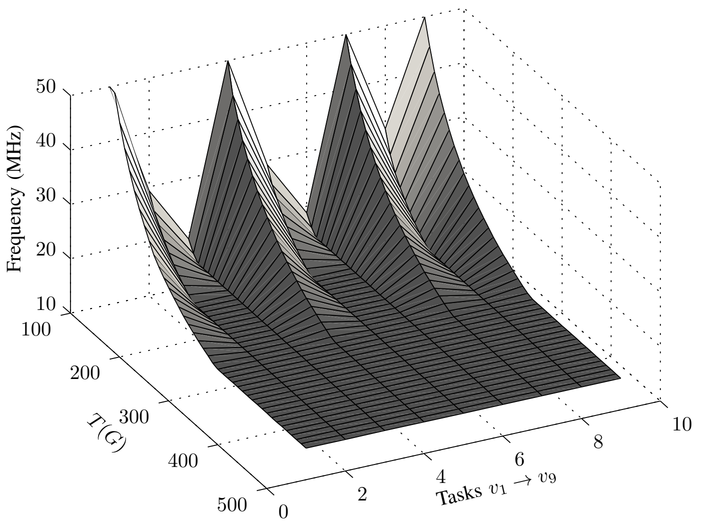

Figure 10. Task frequencies by varying the throughput constraint.

in Figure 1 and the energy to frequency curve illustrated in Figure 5, represented as the minimising objective function (2), we demonstrate how the application's throughput constraint affects the frequencies produced by the convex programming optimisation. We model the application using (4) for the edge constraints. The results of varying the throughput constraint can be seen in Figure 9 for the scenario where all the frequencies are constrained to be the same for the entire application. In the graph there is no data displayed before 100μsec because it is infeasible for the application to finish within its throughput constraint using a maximum frequency of 50 MHz. From this point as we relax the throughput constraint the frequencies for energy consumption minimisation monotonically decrease as is expected using the energy model. The frequency reduction levels off at 15.811 MHz. This corresponds to the frequency associated with the minimum energy in the energy model graph illustrated in Figure 5. Reducing the frequency further would cost more energy per cycle.

Repeating the same experiment again while removing the constraint that all task frequencies must be equal enables the derivation of a frequency level per task, the results of which are displayed in Figure 10. As the throughput constraint relaxes the task frequencies reduce such that all paths are equally critical, i.e. increasing the length of a task anywhere in the graph would increase the graph's iteration period. This is true until the frequencies reach 15.811 MHz where any further reduction in frequency would actually lead to an energy consumption increase.

As is described in Section V platform restrictions such as the application task mapping must be taken into account when applying our technique to an actual platform implementation. For the platform illustrated in Figure 3 we map the application illustrated in Figure 1 onto the platform using the mapping and static ordering illustrated in Figure 7. Carrying out the same experiment as before we now use the edge constraints as described in (6). For this constraint we profile the application and the platform to assign worst case timings for the

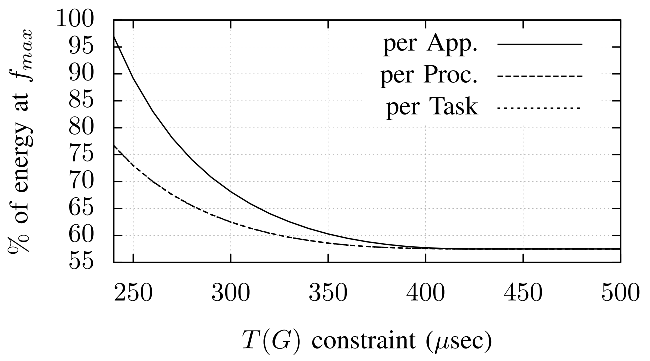

Figure 11. Normalised energy per graph iteration against $T(G)$.

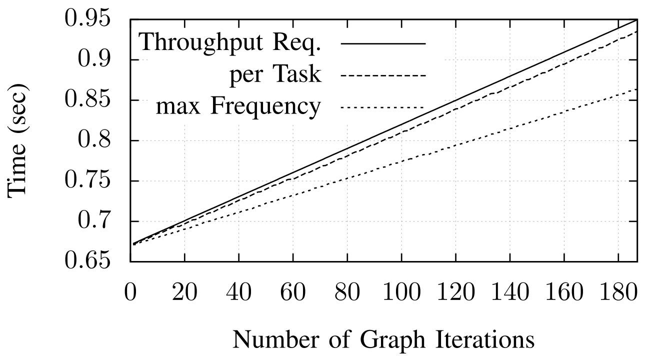

Figure 12. Synthetic graph iteration finishing times.

communication $\phi(v)$ and scheduling $\lambda(v)$ of each task v.

Using the mapping illustrated in Figure 7 we also constrain the frequency derivation to one frequency per μBlaze tile. This produces the energy consumption per graph iteration curve as illustrated in Figure 11. As the throughput constraint is relaxed it can be seen that the energy consumption per iteration decreases due to the lowering of the running frequencies for all the frequency constraint scenarios. In Figure 11 there is no difference between the energy consumption per iteration if the frequencies are derived per task or per processor. This is due to the mapping illustrated in Figure 7 where there are no interprocessor data dependencies during the execution of a single graph iteration, with the exception of v_4 but it is the only task on the processor. Figure 11 illustrates that energy savings of up to 43% may be achieved for the application for this particular configuration. The 43% reduction corresponds with the minimum energy per cycle that is achievable, as may be seen in Figure 5. As such the maximum achievable energy reduction that this technique can provide depends on the availability of static slack, and on the profile of the processor's energy consumption at different frequencies.

For experimentation on an actual MPSoC hardware platform we assign the example application, illustrated in Figure 7, a TDM allocation such that it receives 23.5% processor utilisation and a throughput constraint of 714 iterations per second. Using (7) we calculate that this is equivalent to the

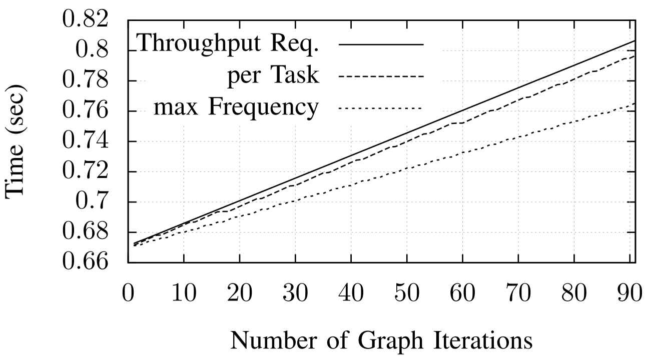

Figure 13. WLAN graph iteration finishing times.

synthetic application graph G_s having a throughput requirement of $T(G_s) = 17500$ cycles per iteration with 100% of the processor resources. We derive the frequencies for this throughput constraint and use (8) to calculate the discrete operating frequencies per task, with our platform supporting 17 equidistant frequency levels from 0 to 50 MHz. The timing results of running the application on an actual implementation of the platform are shown in Figure 12, where it can be seen that the application always meets its throughput requirement. Equation (8) rounds the frequency up to the closest higher discrete frequency level. In Figure 12 the effects of this can be seen as the divergence of the throughput constraint line and the iteration finish line. Without rounding the two lines would continue in a parallel direction. The line representing the finishing time of each iteration of the graph running at f_{max} diverges much more quickly from the throughput requirement line. This divergence is the static slack that we use to perform DVFS and reduce the application's energy consumption while still meeting the application's throughput requirement.

This experiment is similarly carried out for the more complex WLAN application, illustrated in Figure 2, resulting in the graph in Figure 13, which concurs with the behaviour of the technique as observed in Figure 12.

VII. Conclusion

We demonstrate an off-line DVFS technique for dataflow modelled applications that may be mapped onto multiple processing cores of an MPSoC. We show for processors with DVFS power models that may be convexly approximated, how static frequencies may be derived per task to achieve energy reduction. This is achieved through the application of the existing convex programming optimisation technique to an optimisation problem formulated from the processors' DVFS power models, in combination with an application's timed dataflow graph, processor mapping and throughput requirement. We demonstrate that our technique is applicable to an existing real-time MPSoC platform template.

We show experimentally how our technique may be applied to a timed Dataflow graph representation of a WLAN real-time application. Our experimental results confirm that for an actual platform our technique uses the application's static slack to enable the reduction of task frequencies, without violating the application's throughput requirement. For our example DVFS power model that had a 43% energy variation between the highest and lowest energy consuming frequency level, we show that this maximum energy saving is achievable using our technique if permitted by the application's real-time requirements.

References

[1] L. Benini *et al.*, "Allocation, Scheduling and Voltage Scaling on Energy Aware MPSoCs," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 2006.

[2] P. de Langen *et al.*, "Leakage-aware multiprocessor scheduling," *Journal of Signal Processing Systems*, vol. 57, 2009.

[3] H. Kawaguchi *et al.*, "μitron-lp: power-conscious real-time OS based on cooperative voltage scaling for multimedia applications," *IEEE Transactions on Multimedia*, vol. 7, no. 1, 2005.

[4] G. Semeraro *et al.*, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," Feb. 2002.

[5] P. Grosse *et al.*, "Methods for power optimization in SOC-based data flow systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 3, 2009.

[6] E. Lee *et al.*, "Synchronous data flow," *Proc. of the IEEE*, vol. 75, no. 9, 1987.

[7] S. Boyd *et al.*, *Convex Optimization*. Cambridge University Press, 2004.

[8] H. G. Eggleston, *Convexity*. Cambridge University Press, 1958.

[9] R. J. Webster, *Convexity*. Oxford University Press, 1994.

[10] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 1.21," http://cvxr.com/cvx.

[11] A. Molnos *et al.*, "A composable, energy-managed, real-time MPSoC platform," in *OPTIM*, May 2010.

[12] O. Moreira *et al.*, "Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor," ser. EMSOFT '07, 2007.

[13] Y. Shin *et al.*, "Cooperative voltage scaling (CVS) between OS and applications for low-power real-time systems," in *Custom IC's, 2001, IEEE Conference on.*, 2001.

[14] S. Lee *et al.*, "An intra-task dynamic voltage scaling method for SoC design with hierarchical FSM and synchronous dataflow model," ser. ISLPED '02, 2002.

[15] D. Shin *et al.*, "Optimizing intra-task voltage scheduling using data flow analysis," ser. ASP-DAC '05, 2005.

[16] G. Bilsen *et al.*, "Cyclo-static dataflow," *IEEE Trans. on Sig. Proc.*, vol. 44, no. 2, 1996.

[17] K. Goossens *et al.*, "The Aethereal network on chip after ten years: Goals, evolution, lessons, and future," in *Proc. Des. Auto. Con.*, Jun. 2010.

[18] T. Bjerregaard *et al.*, "Implementation of guaranteed services in the MANGO clockless network-on-chip," *Computers and Digital Techniques, IEE Proceedings* -, vol. 153, no. 4, 2006.

[19] M. Millberg *et al.*, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip," in *DATE 2004*, vol. 2, 2004.

[20] Xilinx, website, http://www.xilinx.com.

[21] K. Goossens *et al.*, "Composable dynamic voltage and frequency scaling and power management for dataflow applications," in *DSD 2010*, Sep. 2010.

[22] M. Ekerhult, "CompOSe, design and implementation of a composable and slack-aware operating system targeting a multi-processor system-onchip in the signal processing domain," Master's thesis, Lund University, 2009.

[23] M. Meijer *et al.*, "On-chip digital power supply control for system-on-chip applications," ser. ISLPED '05, 2005.

[24] W. Kim *et al.*, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *HPCA 2008*, 2008.

[25] E. Beigne *et al.*, "Dynamic Voltage and Frequency Scaling Architecture for Units Integration within a GALS NoC," in *NoCS 2008*, 2008.

[26] M. Ichihashi *et al.*, "A 65-nm on-chip multi-mode asynchronous local power supply unit for multi-power domain SoCs achieving fine grain DVS," in *A-SSCC 2009*, 2009.

[27] O. Moreira *et al.*, "Buffer sizing for rate-optimal single-rate data-flow scheduling revisited," *Computers, IEEE Trans. on*, vol. 59, no. 2, 2010.

[28] S. Sriram and S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization, Second Edition*. CRC Press, 2009.