Iterative Schedule Optimisation for Voltage Scalable Distributed Embedded Systems

MARCUS T. SCHMITZ and BASHIR M. AL-HASHIMI
University of Southampton
and
PETRU ELES
Linköping University

We present an iterative schedule optimisation for multi-rate system specifications, mapped onto heterogeneous distributed architectures containing dynamic voltage scalable processing elements (DVS-PEs). To achieve a high degree of energy reduction, we formulate a generalised DVS problem, taking into account the power variations among the executing tasks. An efficient heuristic is presented that identifies optimised supply voltages by not only "simply" exploiting slack time, but under the additional consideration of the power profiles. Thereby, this algorithm minimises the energy dissipation of heterogeneous architectures, including power managed processing elements, effectively. Further, we address the simultaneous schedule optimisation towards timing behaviour and DVS utilisation by integrating the proposed DVS heuristic into a genetic list scheduling approach. We investigate and analyse the possible energy reduction at both steps of the co-synthesis (voltage scaling and scheduling), including the power variations effects. Extensive experiments indicate that the presented work produces solutions with high quality.

Categories and Subject Descriptors: C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems; J.6 [**Computer-aided engineering**]: Computer-aided design

General Terms: Algorithms, Design, Optimization

Additional Key Words and Phrases: Dynamic voltage scaling, Embedded systems, Energy minimisation, Scheduling, System synthesis, Heterogeneous distributed systems

1. INTRODUCTION AND RELATED WORK

The dramatically growing market segment for embedded computing systems is driven by the ever increasing demand for new application specific devices, which can be generally found in almost every application domain, such as consumer electronics, home appliances, automotive, and avionic devices. To help balancing the production costs with development time and cost, these embedded systems are commonly composed of several heterogeneous processing elements (PEs), which are interconnected by communication links (CLs) [Wolf 1994]. For example, very

Authors' addresses: M. T. Schmitz and B. M. Hashimi, Department of Electronics and Computer Science, University of Southampton, SO17 1BJ Southampton, UK, email: {ms99r, bmah}@ecs.soton.ac.uk; P. Eles, Department of Computer and Information Science, Linköping University, S-581 83 Linköping, Sweden, email: petel@ida.liu.se.

often the combination of less powerful and cheap PEs leads to a more cost efficient design implementation than the usage of a powerful single processor system. Typically, such embedded systems have to concurrently perform a multitude of complex tasks under a strict timing behaviour, given in the system specification.

System-level co-design is a methodology aiming to aid the system designers at solving the difficult problem of finding the "best" suitable implementation for a system specification. The traditional co-design flow for distributed systems involves solving three subproblems, namely:

(1) *Allocation:* determining the numbers and types of PEs and CLs used to compose the system architecture,
(2) *Mapping:* assignment of computational tasks to PEs and of data transfers between different PEs to CLs,
(3) *Scheduling:* determining the execution order (sequencing) of tasks mapped to a PE and communications mapped to a CL.

These problems (allocation/mapping and scheduling) are well-known to be NP-complete [Garey and Johnson 1979], and therefore an optimal co-design of distributed systems is intractable. This justifies the usage of heuristic optimisation algorithms of different types, e.g., simulated annealing [Henkel et al. 1993; Eles et al. 1997], tabu-search [Eles et al. 1997], genetic algorithm [Dick and Jha 1998; Teich et al. 1997], or constructive techniques [Wolf 1997], to tackle the computational complexity.

In the last decade power dissipation has become a mandatory issue of concern in the design of embedded systems because of: (a) The popularity of mobile applications powered by batteries with limited capacity, (b) the operational costs and environmental reasons affected by the high electrical power consumption of large computing systems, and (c) the reliability and feasibility problems caused by extensive heat production exceeding the physical substrate limitations, especially when implementing systems on a single chip (SoCs). Several useful techniques have been proposed to reduce the power dissipation of integrated circuits, targeted at different levels of abstraction [Devadas and Malik 1995; Pedram 1996]. One approach aiming to reduce the power dissipation at the system-level is recently receiving a lot of attention from the research community and industry, namely, dynamic voltage scaling (DVS) [Weiser et al. 1994; Gutnik and Chandrakasan 1997; Hong et al. 1999; Ishihara and Yasuura 1998; Okuma et al. 1999; Quan and Hu 2001; Shin and Choi 1999; Shin et al. 2000; Simunic et al. 2001]. The main idea behind DVS is to conjointly scale the supply voltage V_{dd} and operational frequency f dynamically during run-time in accordance to the temporal performance requirements of the application. In this way the dynamic power dissipation P_{dyn} (disregarding short-circuit power) is reduced in a near cubic manner, since it depends quadratically on the supply voltage and linearly on the operational frequence. The exact relation is expressed by the following two equations,

$$P_{dyn} = C_L \cdot N_{0\to1} \cdot f \cdot V_{dd}^2 \tag{1}$$

$$f = k \cdot \frac{(V_{dd} - V_t)^2}{V_{dd}} \tag{2}$$

where C_L denotes the load capacitance of the digital circuit, $N_{0\rightarrow1}$ represents the zero to one switching activity, k is a circuit dependent constant, and V_t is the threshold voltage. DVS is thereby able to exploit the idle and slack times (time intervals where system components do not carry out any computations), given in the system schedule, in order to lower the power dissipation. The occurrence of idle and slack times has three reasons: (a) It is often the case for a given application to show various degrees of parallelism, i.e., not all PEs will be utilised constantly during run-time, (b) the performance of the allocated architecture cannot be adapted perfectly to the application needs, since the allocation of "performance" is not given as continuous range, but is rather quantised, and (c) schedules for hard real-time systems are constructed by considering worst case execution times (WCETs), however, actual execution times of tasks during operation are, for most of their activations, smaller than their WCETs. Several state-of-the-art implementations of DVS enabled processors [Burd et al. 2000; Gutnik and Chandrakasan 1997; Klaiber 2000] have successfully shown that power consumption can be reduced significantly (by up to 10 times compared to fixed voltage approaches) when running real world applications. In order to achieve such a high level of power and energy efficiency, it is essential to identify optimised scaling voltages for the task executions [Okuma et al. 2001] to exploit the available idle and slack times efficiently. Such voltage scheduling algorithms can be divided in two broad categories: on-line (dynamic) [Lee and Sakurai 2000; Quan and Hu 2001; Shin and Choi 1999] and off-line (static) approaches [Bambha et al. 2001; Gruian and Kuchcinski 2001; Ishihara and Yasuura 1998]. The first class dynamically re-calculates the priorities and scaling voltages of tasks at run-time, i.e., the voltage schedule is changed during the execution of the application. Obviously, such approaches consume additional power and time during execution. On the other hand, they are able to make use of the dynamic slack introduced by execution times smaller than the WCET. In the second class, a static voltage schedule is calculated once before the application is executed, i.e., the voltage schedule is maintained unchanged during run-time. Hence, power and time overheads are avoided. The technique proposed in this paper falls into the class of static voltage schedulers.

Voltage selection is already a complex problem when only single DVS processor systems, executing *independent* tasks, are considered [Hong et al. 1999]. The problem is further complicated in the presence of distributed systems specified by *dependent* tasks where the allocation, mapping, and scheduling influence the possibility to exploit DVS [Bambha et al. 2001; Gruian 2000; Luo and Jha 2000; Schmitz and Al-Hashimi 2001]. Most previous DVS approaches [Hong et al. 1999; Lee and Sakurai 2000; Quan and Hu 2001; 2002; Shin and Choi 1999] concentrate on single processor systems executing *independent* task sets and, hence, are not directly applicable to the problem addressed here. Nevertheless, we need to consider DVS at all these optimisation steps during co-synthesis, in order to find high quality system implementations. In this paper, we will concentrate on the scheduling and voltage scaling aspects of such systems. Further details concerning the mapping and allocation steps can be found in [Schmitz et al. 2002; Schmitz 2003].

Previous research in system-level co-synthesis is extensive but has mainly focused on traditional architectures *excluding* issues related to power consumption [Ernst

et al. 1993; Henkel and Ernst 2001; Micheli and Gupta 1997; Prakash and Parker 1992; Wolf 1997; Xie and Wolf 2001] or considering energy optimisation with components that are *not* DVS enabled [Dick and Jha 1998; Kirovski and Potkonjak 1997]. A system-level scheduling technique for power-aware systems in mission-critical applications was presented in [Liu et al. 2001]. This approach satisfies min/max timing constraints as well as max power constraints taking into account not only processor power consumption but additionally the power dissipated by peripheral system components. All this research provides a valuable basis for the work presented here. However, three research groups recently proposed approaches for the voltage scaling problem in distributed systems that have close relationship to the problems we address in this paper. Bambha [Bambha et al. 2001] presented a hybrid search strategy based on simulated heating. This method uses a global genetic algorithm to find appropriate parameter settings for a local search algorithm. The local search algorithms are based on hill climbing and Monte Carlo techniques.

In [Luo and Jha 2000], a power conscious joint scheduling of aperiodic and periodic tasks is introduced, which reserves execution slots for aperiodically arriving tasks within a static schedule of a task graph. Their algorithm aims for energy minimisation through DVS by distributing the available deadline slack evenly among all tasks. They further extend their approach towards a battery-aware scheduling with the aim to improve the battery discharge profile [Luo and Jha 2001]. Gruian and Kuchcinski [Gruian and Kuchcinski 2001] extend a dynamic list based scheduling heuristic to support DVS by making the priority function energy aware. In each scheduling step the energy sensitive task priorities are re-calculated. If a scheduling attempt fails (exceeded hard deadline), the priority function is adjusted and the application is re-scheduled. Despite their power reduction efficiency, all these DVS approaches [Bambha et al. 2001; Gruian and Kuchcinski 2001; Luo and Jha 2000] do not consider and target heterogenous distributed architectures containing *power managed DVS-PEs* in which the dissipated power for each task execution might vary. It was shown in [Ishihara and Yasuura 1998] and [Manzak and Chakrabarti 2000] that the variations in the average switching activity (equivalent to variations in power) influence the optimal voltage schedule and hence need to be considered during the voltage selection. However, both approaches do not target distributed systems with multiple PEs executing tasks with dependencies. Thus, new system-level co-synthesis approaches for DVS-enabled architectures, which take into account that power *varies* among the executed tasks, are needed. Recently, an approach to solve this problem has been presented in [Zhang et al. 2002]. The scheduling optimisation towards DVS utilisation in this approach is based on a constructive technique, as opposed to our iterative scheduling optimisation which allows a thorough search to find schedules of high quality.

In this paper, we formulate a generalised DVS problem that considers power variation effects and is based on an iterative scheduling optimisation. We assume that typical embedded architecture employ gate level power reduction techniques, such as gated clocks, to switch off un-utilised blocks in the circuit [Devadas and Malik 1995; Tiwari et al. 1994]. It is therefore necessary to take into account that power varies considerably among the tasks carried out by the system. This holds also for DVS-PEs [Burd 2001]. For example, in the case of a general purpose

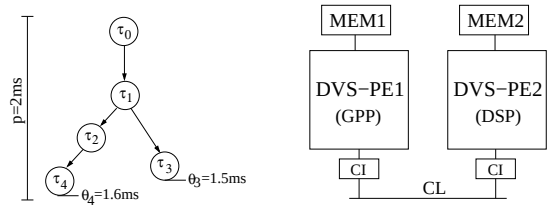
processor (GPP) including an integer and a floating point unit, it is not desirable to keep the floating point unit active if only integer instructions are executed. Thereby, different tasks (different use of instructions) dissipate different amounts of power on the same PE. In the case of an ARM7TDMI processor the current varies between 5.7 and $18.3mA$, depending on the functionality which is carried out [Brandolese et al. 2000]. Taking this into account, the assumptions to Lemma 1 and Lemma 2 in [Ishihara and Yasuura 1998], stating that energy consumption is independent from the type of operations and input data and depends only on the supply voltage, have to be rejected. In addition to this, our problem formulation also takes into consideration the different power dissipations among different processing elements. This is important since high power consuming PEs are likely to have a greater impact on the energy saving (when scaled to lower performance) than low power consuming PEs.

The aim of this paper is twofold: Firstly, we are formulating and examining a generalised DVS problem which allows power variations, in the following also called PV-DVS problem. We introduce a new, generalised DVS heuristic for distributed systems containing heterogenous and power managed PEs. Secondly, we illustrate the incorporation of this scaling technique into a genetic list scheduling approach, which optimises the system schedule simultaneously towards timing feasibility and DVS exploitability. This incorporation necessitates a careful adaption of the employed list scheduler to ensure its suitability for both optimisation goals. We provide a detailed analysis of the DVS and scheduling approach revealing how scheduling influences the DVS utilisation. This analysis is carried out on several benchmark examples from literature [Bambha et al. 2001; Gruian and Kuchcinski 2001] and generated for experimental purposes, as well as on an optical flow detection real-life examples.

The remainder of the paper is organised in the following way. In Section 2, we formulate the system-level synthesis problem and give a general and brief overview of genetic algorithms, since they are used for the schedule optimisation. Section 3 describes in detail our approach to the system-level scheduling problem for architectures including DVS components. In Section 4 numerous benchmark examples are evaluated and compared with approaches that neglect power profile information. Finally, in Section 5 we give some conclusions drawn from the presented work.

2. PROBLEM FORMULATION AND PRELIMINARIES

In this work, we consider that a multi-rate application is specified as a set of communicating tasks, represented by a task graph $G_S(\mathcal{T}, \mathcal{C})$. This (hyper) task graph might be the combination of several smaller task graphs, capturing all task activations for the hyper-period (LCM of all graph periods). Figure 1(a) shows a task graph example. Each node $\tau \in \mathcal{T}$ in these acyclic directed graphs represents a task, an atomic unit of functionality to be executed without preemption. Further, each task might inherit a specific hard deadline θ. These deadlines must be met to fulfil the feasibility requirements of the specified application. In addition, the task graph inherits a period p which specifies the maximal allowed time between two successive invocations of the initial task. The edges $\gamma \in \mathcal{C}$, in the task graph, denote precedence constraints and data dependencies between tasks. If two tasks,



(a) Single task graph example with specified deadlines

(b) Architecture containing two DVS-PEs with local memory linked by a single bus

Fig. 1. Specification and Architectural Models

τ_i and τ_j, are connected by an edge then the execution of task τ_i must be finished before task τ_j can be started. Data dependencies inherit a data value, reflecting the quantity of information to be exchanged by two tasks. A feasible implementation of an application must respect all timing constraints and precedence requirements when executed on an underlying architecture. This type of specification model is most suitable for data flow intensive application with a repetitive behaviour, as they can be found in systems for image, speech, and video processing.

The architectures we consider here consist of heterogeneous PEs, like general purpose processors (GPPs), ASIPs, FPGAs, and ASICs. These components include state-of-the-art DVS-PEs. An infrastructure of communication links, like buses and point-to-point connections, interconnects these PEs. Processors are capable to execute software tasks, which are accommodated in local memory, in a sequential manner, while tasks implemented on FPGAs or ASICs can be performed in parallel and occupy silicon area. Figure 1(b) shows an example architecture built out of two DVS-PEs connected by a single bus. Such architectures can be found in application domains which target multimedia and telecommunication systems. The architecture is captured using a directed graph $G_A(\mathcal{P}, \mathcal{L})$ where nodes $\pi \in \mathcal{P}$ represent processing elements and edges $\lambda \in \mathcal{L}$ denote communication links.

Each task of the system specification might have multiple implementation alternatives and can therefore be potentially mapped to several PEs able to execute this task. If two communicating tasks are accommodated on different PEs, π_n and π_m with $n \neq m$, then the communication takes place over a CL, involving a communication time overhead. For each possible task mapping certain implementation properties, like e.g. execution time, dynamic power dissipation, memory, and area requirements, are given in a technology library. These values are either based on previous design experience or on estimation and measurement techniques [Brandolese et al. 2000; Fornaciari et al. 1999; Li et al. 1995; Tiwari et al. 1994; Muresan and Gebotys 2001]. The technology library further includes information about the available PEs and CLs, such as price, DVS enable flag, etc.

The overall co-synthesis process includes three traditional co-design problems, namely, *allocation, mapping,* and *scheduling*. These optimisation steps determine

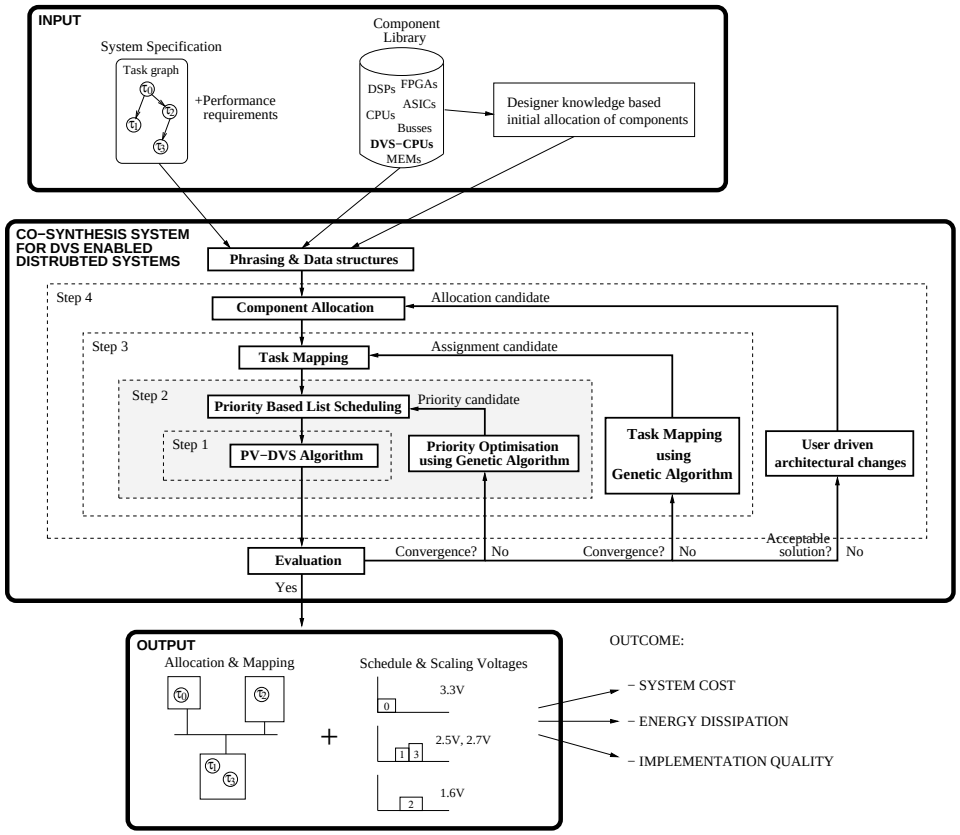


Fig. 2. Design flow of our generalised PV-DVS co-synthesis methodology for heterogeneous and power managed processing elements

an architecture, a possible mapping of the functionality over the components of that architecture, and a feasible schedule, such that certain design objectives are either minimised or maximised. However, the problem, as presented here, involves an additional *voltage scaling* step, which identifies scaling voltages for the tasks mapped to DVS-PEs, in order to minimise the energy dissipation. Obviously, due to their interrelations these design decisions cannot be made independently of each other. In this paper, we will particularly focus on the scheduling and voltage scaling problem in such DVS enabled architectures.

3. SYSTEM SYNTHESIS APPROACH INCLUDING POWER MANAGED PES

In this section, we give an overview of the co-synthesis flow for distributed architectures containing heterogeneous, power managed processing elements. The presented approach is based on a generalised power model, which allows to capture power variation effects among the tasks (the power profiles). The whole design flow is illustrated in Figure 2. As mentioned earlier, this paper concentrates on the DVS algorithm and the scheduling technique, indicated as Steps 1 and 2. However, we give a brief overview of our system-level synthesis approach. The input

information consists of two main parts, a system specification and a component library. The system specification is captured using the directed acyclic graph, as outlined in Section 2, and includes the performance requirements. The properties of processing elements and communication links (price, idle power dissipation, etc.) are collected in the component library, which additionally includes estimated information (i.e. execution time, dynamic power dissipation, etc.) about each task/PE and communication/CL combination. The input to our co-design approach further involves a knowledge-based pre-allocation of system components. Using the presented synthesis approach the designer evaluates the suitability of this allocation. If an architecture proves to be unsuitable or of low quality the designer modifies the allocation and re-evaluates the design. The presented co-synthesis system takes this input information and establishes the necessary data structures. This is followed by the allocation step (Step 4 in Figure 2) that determines the type and quantity of PEs and CLs used to compose the architecture. An appropriate allocation minimises system cost while providing sufficient computational performance. In Step 3, the task mapping is carried out. This step determines the mapping of tasks to PEs and uses a GA-based iterative improvement technique. Task mapping optimises the distribution of tasks towards energy savings, but additionally aims to satisfy imposed area constraints on hardware components. After a mapping is established, the next step involves the scheduling (Step 2) of the tasks and communications in order to meet the hard time constraints of the application and to further minimise the energy dissipation in the presence of DVS-PEs. This optimisation is based on a list scheduling heuristic using a GA for the determination of priorities. At the core of this co-synthesis approach, as shown in Figure 2, is the PV-DVS algorithm (Step 1). In this step the algorithm identifies scaling voltages for the task executions on DVS-PEs under the consideration of power variations in order to efficiently reduce the energy dissipation of the distributed system. The output of the proposed co-design flow consists of three results: (a) an allocated architecture, (b) a mapping of tasks and communications onto that architecture, and (c) a feasible schedule for the task executions and the communication activities, such that no time constraints are violated. In addition to these traditional aspects, the proposed technique further outputs scaling voltages for the tasks executed by DVS-PEs. Note that the architecture, the mapping, and the schedule are optimised for the exploitation of DVS and therefore differ from the results obtained by traditional co-design approaches [Dick and Jha 1998; Ernst et al. 1993; Kirovski and Potkonjak 1997; Micheli and Gupta 1997; Prakash and Parker 1992; Wolf 1997]. The outcome, which is of relevance to the designer/architect, consists of the system cost (system price), the total system energy dissipation, and the implementation quality (e.g. performance related to soft deadlines). Using these values, the designer is able to judge the overall quality of the implementation and can operate certain changes if necessary.

3.1 Generalised DVS Approach for Distributed Systems containing Power Managed PEs

In this section, which is concerned with the identification of scaling voltages, we first motivate the consideration of power variation effects using an illustrative example. This is followed (Subsection 3.1.1) by the formulation of a generalised DVS problem for distributed systems. In Subsection 3.1.2, we introduce an heuristic algorithm

to solve the formulated problem.

The aim of the generalised DVS approach is to identify scaling voltages under the consideration of power variation effect. This is done for the scheduled and mapped system specification such that the total dynamic energy dissipation is minimised. The presented approach assumes that no restrictions are placed on the scaling voltages, i.e., our technique targets variable-voltage systems (nearly continuous range of possible supply voltages) rather than multi-voltage systems (small and limited number of potential supply voltages). However, we will explain in Section 3.1.2 how the obtained scaling voltages can be easily adapted to suit multi-voltage systems. The term *generalised DVS* refers to the key observation that the power dissipation varies considerably upon the PE types and the instructions executed by the PEs. This is not new and well known [Burd and Brodersen 1996; Tiwari et al. 1994]. However, unlike previous approach to DVS for distributed systems [Bambha et al. 2001; Gruian and Kuchcinski 2001; Luo and Jha 2000], the presented technique takes these power variation effects into account and is sufficiently fast to be used in the inner optimisation loop of a co-synthesis tool. The following example is used to motivate the necessity of considering power variations during the voltage selection, in order to minimise the dynamic energy dissipated by the system. Before we start with the example, it is necessary to define the term *energy difference*, which will be used throughout this section.

Definition 1. We define an energy difference ΔE_τ as the difference between the energy dissipation of task τ with the execution time t and the reduced energy dissipation (due to voltage and clock scaling) of the same task when extended by a time quantum Δt. Formally:

$$\Delta E_\tau = E_\tau(t) - E_\tau(t + \Delta t) \tag{3}$$

where $E_\tau(t)$ and $E_\tau(t + \Delta t)$ are calculated using Equations (1) and (2). □

Motivational Example 1: Considering Power Variations during Voltage Scaling

The intention with this illustrative example is to motivate the consideration of power variation effects during the voltage scaling of heterogeneous distributed systems. This is done by using two different models during the voltage scaling: (a) a fixed power model which does not allow power variations and (b) a variable power model which takes power variation into account (as used in the proposed approach).

The starting point for the DVS technique is a system specification scheduled (at nominal voltage) and mapped onto an allocated architecture which includes power managed DVS components. In this simple example, we consider an architecture composed of two hypothetical, heterogeneous DVS-PEs connected through a single bus as illustrated in Figure 1(b). The system is specified by the task graph shown in Figure 1(a).

Nominal supply voltage V_{max} and threshold voltage V_t for the two PEs are given in Table II(a). This table further shows the nominal execution times and dynamic power dissipations of tasks, according to their mapping. Furthermore, the transfer times and power dissipations of the communication activities are shown in Table II(b), reflecting the inter PE communications through the bus. Communications between tasks on the same PE are assumed to be instantaneous, and their

Table I. Execution times and power dissipations for the motivational example

task	**PE0** ($V_{max} = 5V$, $V_t = 1.2V$)		**PE1** ($V_{max} = 3.3V$, $V_t = 0.8V$)	
	exe. time (ms)	*power dis.* (mW)	*exe. time* (ms)	*power dis.* (mW)
τ_0	0.15	85	0.70	30
τ_1	0.40	90	0.30	20
τ_2	0.10	75	0.75	15
τ_3	0.10	50	0.15	80
τ_4	0.15	100	0.20	60

(a) Task execution times and power dissipations at nominal supply voltage

comm.	*comm. time* (μs)	*power dis.* (mW)
$\gamma_{0\to1}$	0.05	5
$\gamma_{1\to2}$	0.05	5
$\gamma_{1\to3}$	0.15	5
$\gamma_{2\to4}$	0.10	5

(b) Communication times and power dissipations of communication activities mapped to the bus

power dissipation is neglected, as in most co-synthesis approaches.

A possible mapping and scheduling of the system tasks onto the underlying architecture is shown in Figure 3, which describes the power dissipation over time, hence, the power profile of PEs and CLs. It can be observed that PE0 accommodates tasks τ_0 and τ_4, while the remaining tasks are mapped to PE1. The communication link, connecting both PEs, shows two communications, $\gamma_{0\to1} = (\tau_0, \tau_1)$ and $\gamma_{2\to4} = (\tau_2, \tau_4)$. The dynamic system energy dissipation of this configuration at nominal supply voltage can be calculated as $57.75\mu J$, using the dynamic power values and execution times given in Tables II(a) and II(b). Obviously, since the execution of task τ_3 finishes at $1.4ms$ and the task deadline is at $1.5ms$, a slack of $0.1ms$ is available, as indicated in Figure 3. The same holds for task τ_4, which finishes its execution after $1.5ms$, leaving a slack of $0.1ms$ until the deadline is reached. These slacks can be used to extend the task execution times. Thus, the DVS-PEs can be slowed down by scaling the supply voltage and accordingly the clock frequency, following the relation given in Equation (2). Let us consider two cases for the identification of scaling voltages: (a) When a fixed power model is used (power variations are neglected), i.e., all tasks mapped to the same PE are assumed to consume the same constant amount of power, and (b) a more generalised and more realistic power model allowing for power variations among the tasks (as proposed in this work).

One approach to optimise the energy dissipation, which neglects the power pro-

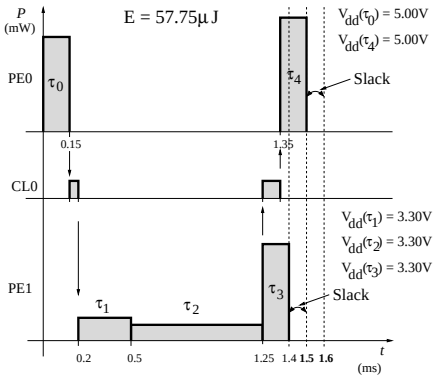


Fig. 3. Power profile of a possible mapping and schedule at nominal supply voltage (no DVS is applied)

file, is to distribute the slack time evenly among the tasks. This is illustrated in Figure 4(a) where each task execution is extended using a factor of $e = 1.45/1.35 = 1.074$, since the total computation time (not including communications) accounts for $1.35ms$ and the available time (given the slack of $0.1ms$) is $1.45ms$. Thereby, the extended task executions can be calculated as the following: $t_0 = 0.161ms$, $t_1 = 0.322ms$, $t_2 = 0.856ms$, $t_3 = 0.161ms$, and $t_4 = 0.161ms$. This allows to lower the supply voltages of PE0 and PE1 to $4.79V$ and $3.16V$, respectively, according to following equation (derived from Equation (2)).

$$V_{dd} = V_t + \frac{V_0}{2d^*} + \sqrt{\left(V_t + \frac{V_0}{2d^*}\right)^2 - V_t^2} \tag{4}$$

where d^* denotes the normalised delay, which, in this example, is equal to the extension factor e. The constant V_0 is given by:

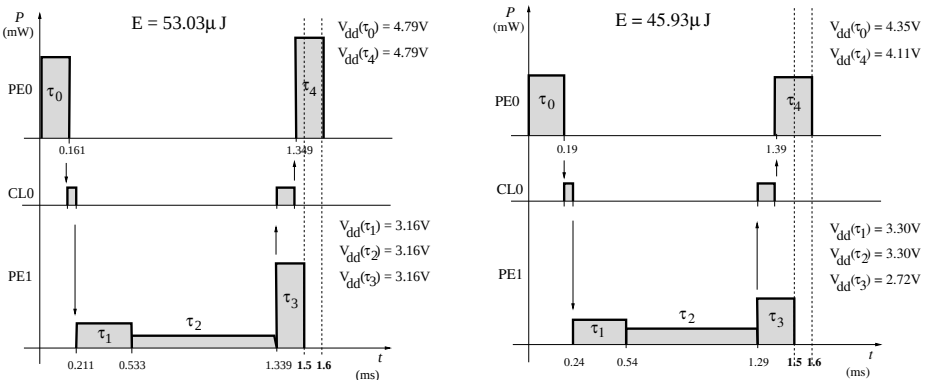
$$V_0 = \frac{(V_{max} - V_t)^2}{V_{max}} \tag{5}$$

Thus, the reduced voltages of PE0 is calculated as:

$$\begin{aligned} V_{dd} &= 1.2V + \frac{(5V - 1.2V)^2/5V}{2 \cdot 1.074} + \sqrt{\left(1.2V + \frac{(5V - 1.2V)^2/5V}{2 \cdot 1.074}\right)^2 - (1.2V)^2} \\ V_{dd} &= 4.788V \end{aligned}$$

using the nominal supply voltage $V_{max} = 5V$ and the threshold voltage $V_t = 1.2V$ as given in Table II(a). In the same way, the scaled supply voltage for PE1 can be calculated as $V_{dd} = 3.161V$, using $V_{max} = 3.3V$ and $V_t = 0.8V$. Adjusting the supply voltages of the PEs to theses levels, the task deadlines are still satisfied, and the power dissipations are reduced. According to Equation 1, the power dissipation of each task can be calculated using the following relation:

$$\frac{P_{V_{dd}}}{P_{V_{max}}} = \frac{\alpha \cdot C_L \cdot f_{V_{dd}} \cdot V_{dd}^2}{\alpha \cdot C_L \cdot f_{V_{max}} \cdot V_{max}^2} = \frac{1}{e} \cdot \frac{V_{dd}^2}{V_{max}^2} = \frac{1}{d^*} \cdot \frac{V_{dd}^2}{V_{max}^2} \tag{6}$$



(a) DVS neglecting the power profile (traditional approach for voltage scaling of dependent tasks); reducing energy dissipation by 8.2%

(b) DVS using a generalised power model (allowing for variations and the proposed energy difference approach; reducing energy dissipation by 20.5%%

Fig. 4. Power profiles of the schedules when DVS is applied, resulting in reduced energy dissipations

considering that $\alpha \cdot C_L$ is constant for a given task and that relation between reduced operational frequency $f_{V_{dd}}$ and maximal operational frequency $f_{V_{max}}$ is equivalent to the inverse of the extension factor $1/e = f_{V_{dd}}/f_{V_{max}}$. Thereby, the power dissipations are reduced to $P_0 = 72.57mW$, $P_1 = 17.08mW$, $P_2 = 12.81mW$, $P_3 = 68.33mW$, and $P_4 = 85.38mW$. This results in a total energy dissipation $E = 53.03\mu J$, a reduction of 8.2%.

Now consider the case when the generalised power model (allowing power variations) is employed during the identification of scaling voltages for the task executions. This optimisation is based on an energy difference as defined in Equation (3). For a simpler illustration of the method, the available deadline slack is divided into 10 time quanta with the size of $0.01ms$ each. Having defined the time quantum size Δt, we can now calculate an energy difference ΔE_τ for each task, using Equations (1), (2), and (3). For instance, the energy dissipation of task τ_0 at nominal supply voltage can be calculate as $E_0(0.15) = 0.15ms \cdot 85mW = 12.75\mu J$, using the values given in Table II(a). An extension of $0.01ms$ leads to an dissipated energy of $E_0(0.16) = 0.16ms \cdot 73.69mW = 11.79\mu J$ and thereby results in an energy difference $\Delta E_0 = 12.75\mu J - 11.79\mu J = 0.96\mu J$. In the same way, the energy differences can be calculated for the remaining task as $\Delta E_1 = 0.234\mu J$, $\Delta E_2 = 0.156\mu J$, $\Delta E_3 = 0.899\mu J$, and $\Delta E_4 = 1.130\mu J$. Certainly, the task with the highest energy difference (task τ_4) will improve the energy dissipation by the highest amount when extended by Δt. Based on this observation, all remaining time quanta are iteratively distributed among the tasks and the energy difference of extended tasks are recalculated since the energy versus execution time function is non-linear (see Equations (1), (2) and (3)). This optimisation process is illustrated through Table II. Each line in this table corresponds to an iteration and shows

Table II. Energy differences during the execution of the PV-DVS algorithm

	Energy difference ΔE (μJ)				
iteration	τ_0	τ_1	τ_2	τ_3	τ_4
1	0.960	0.234	0.156	0.899	**1.130**
2	0.960	0.234	0.156	0.899	**0.965**
3	**0.960**	0.234	0.156	0.899	0.833
4	0.820	0.234	0.156	**0.899**	0.833
5	0.820	0.234	0.156	0.768	**0.833**
6	**0.820**	0.234	0.156	0.768	0.725
7	0.708	0.234	0.156	**0.768**	0.725
8	0.708	0.234	0.156	0.663	**0.725**
9	**0.708**	0.234	0.156	0.663	0.636
10	0.616	0.234	0.156	**0.663**	0.636
11	0.616	0.234	0.156	0.578	**0.636**
12	**0.616**	0.234	0.156	0.578	0.562
13	0.541	0.234	0.156	**0.578**	0.562
14	0.541	0.234	0.156	0.507	**0.562**
15	-	-	-	**0.507**	-
16	-	-	-	**0.451**	-
extension	4	0	0	6	6

the extendable tasks and their potential energy gain. The bold numbers indicate which task is extended in each iteration, and in this simple example task τ_4 is the first task to be extended (iteration 1). Observing iteration 2, the energy difference of task τ_4 has changed to $\Delta E_4 = 0.965\mu J$, however, τ_4 is still the task which will gain most from an extension. This iterative extension of tasks is repeated until no slack is left. The last row of Table II shows how many extensions are distributed to each task. Accordingly, the new execution times are as follows: $t_0 = 0.19ms$, $t_1 = 0.3ms$, $t_2 = 0.75ms$, $t_3 = 0.21ms$, and $t_4 = 0.21ms$. These extended execution times allow to lower the supply voltages, which results in the following power dissipations: $P_0 = 50.77mW$, $P_1 = 20mW$, $P_2 = 15mW$, $P_3 = 38.74mW$, and $P_4 = 48.33mW$. The total energy dissipation is $E = 45.93\mu J$. This means an energy reduction of 20.5% compared to a reduction of 8.2% obtained with a power profile neglecting approach. □

3.1.1 *Generalised DVS Problem Formulation.* The DVS problem, including power variation effects, can be stated as follows:

> Find for all DVS-PE mapped tasks $\tau \in \mathcal{T}_{\text{DVS}}$ of the system specification a single scaling voltage $V_{dd}(\tau)$ (between the threshold voltage V_t and the nominal supply voltage V_{max}) under consideration of individual power dissipations $P_{max}(\tau)$ such that the dynamic energy dissipation E_Σ is minimised and no deadline and precedence constraints are violated.

The problem can be mathematically expressed using the following definitions, where $\mathbb{R}_0^+ = \{x \,|\, x \in \mathbb{R},\, 0 \leq x < +\infty\}$ and $\mathbb{R}^+ = \mathbb{R}_0^+ \setminus 0$:

—$G_S(\mathcal{T}, \mathcal{C})$ is the system specification graph, where $\mathcal{T}$ is the set of tasks and $\mathcal{C}$ is the set of communications, as defined in Section 2

—$G_A(\mathcal{P}, \mathcal{L})$ is a directed architecture graph, where $\mathcal{P}$ is the set of PEs and $\mathcal{L}$ is the set of CLs, as defined in Section 2

—$\mathcal{P}_{DVS} \subseteq \mathcal{P}$ denotes the set of all DVS-enabled processing elements

—$\mathcal{A} = \mathcal{T} \cup \mathcal{C}$ defines the set of all activities

—$\mathcal{K} = \mathcal{P} \cup \mathcal{L}$ defines the set of all allocated components

—$\mathcal{T}_{\text{DVS}} \subseteq \mathcal{T}$ denotes the set of all tasks mapped to DVS-PEs $\mathcal{P}_{DVS}$

—$P_{max} : \mathcal{T} \mapsto \mathbb{R}^+$ is a function returning the power dissipation of task τ executed at maximal PE supply voltage V_{max}

—$t_{min} : \mathcal{T} \mapsto \mathbb{R}^+$ is a function returning the minimal execution time of task $\tau \in \mathcal{T}$ at maximal PE supply voltage V_{max}

—$V_t : \mathcal{P} \mapsto \mathbb{R}^+$ is defined as a function which returns the threshold voltage of the PE to which task $\tau \in \mathcal{T}$ is mapped

—$V_{max} : \mathcal{T} \mapsto \mathbb{R}^+$ is a function returning the maximal supply voltage of the PE to which task $\tau \in \mathcal{T}$ is mapped

—$\mathcal{T}_d \subseteq \mathcal{T}$ denotes the set of all tasks having a hard deadline

—$t_{exe} : \mathcal{A} \mapsto \mathbb{R}^+$ is a function defined by:

$$t_{exe} = \begin{cases} t_{min}(\epsilon) \cdot \frac{V_{dd}(\epsilon)}{(V_{dd}(\epsilon)-V_t(\epsilon))^2} \cdot \frac{(V_{max}(\epsilon)-V_t(\epsilon))^2}{V_{max}(\epsilon)} & \text{if } \epsilon \in \mathcal{T} \\ t_C & \text{if } \epsilon \in \mathcal{C} \end{cases}$$

where t_C is the communication time for the communication activity $\gamma \in \mathcal{C}$

—$t_d : \mathcal{T}_d \mapsto \mathbb{R}_0^+$ is a function returning the deadline of task $\tau \in \mathcal{T}_d$

—$\mathcal{C}^{in} : \mathcal{T} \mapsto 2^{\mathcal{C}}$ returns the set of all ingoing edges of task $\tau \in \mathcal{T}$

—$t_S : \mathcal{A} \mapsto \mathbb{R}_0^+$ is a function which returns the start time of an activity $\epsilon \in \mathcal{A}$ (i.e., the time when the activity begins execution)

—$A : \mathcal{K} \mapsto 2^{\mathcal{A}}$ defines a function, returning the set of all activities mapped to a component $\kappa \in \mathcal{K}$

—$\mathcal{I} = [t_S(\epsilon), (t_S(\epsilon) + t_{exe}(\epsilon))]$ is the execution interval of activity $\epsilon \in \mathcal{A}$

—$i : \mathcal{A} \mapsto \mathbb{R}_0^+ \times \mathbb{R}_0^+$ is a function returning the execution interval of an activity $\epsilon \in \mathcal{A}$

Using this definitions it is possible to formalise the problem mathematically as the minimisation of

$$E_\Sigma = \sum_{\tau \in \mathcal{T}_{\text{DVS}}} P_{max}(\tau) \cdot t_{min}(\tau) \cdot \frac{V_{dd}^2(\tau)}{V_{max}^2(\tau)}$$

subject to

$$V_t(\tau) < V_{dd}(\tau) \leq V_{max}(\tau),\ \forall \tau \in \mathcal{T}_{\text{DVS}}$$

$$t_S(\tau) + t_{exe}(\tau)\ \leq\ t_d(\tau),\ \forall \tau \in \mathcal{T}_d$$

$$t_S(\gamma) + t_{exe}(\gamma)\ \leq\ t_S(\tau),\ \forall \tau \in \mathcal{T},\ \gamma \in \mathcal{C}^{in}(\tau)$$

$$i(\epsilon_n) \cap i(\epsilon_m) = \varnothing,\ \forall\ (\epsilon_n, \epsilon_m)\ \text{ so that }\ \epsilon_n \in A(\kappa_1), \epsilon_m \in A(\kappa_2) \Rightarrow \kappa_1 = \kappa_2$$

Please note that a single scaling voltage for each task executing on a DVS-PE has to be calculated for the statically scheduled application. However, in dynamically

Algorithm: ***PV_DVS_OPTIMISATION***

Input: - task graph $G_S(\mathcal{T}, \mathcal{C})$, mapping, schedule, architectural information, minimum extension time Δt_{min}
Output: - energy optimised voltages $V_{dd}(\tau)$, dissipated dynamic energy E

01: Generate MSTG from G_S
02: $\mathbb{Q}_E \leftarrow \varnothing$
03: **for** all $(\tau \in \mathcal{T}_d)$ $\{\Delta t_d(\tau) := t_d(\tau) - (t_S(\tau) + t_{exe}(\tau))\}$
04: **for** all $(\tau \in \mathcal{T})$ {calculate t_ϵ}
05: **for** all $(\tau \in \mathcal{T})$ {**if** $t_\epsilon \geq \Delta t_{min}$ **then** $\mathbb{Q}_E := \mathbb{Q}_E + \tau$}
06: $\Delta t = \frac{\min t_\epsilon}{|\mathbb{Q}_E|}$, **if** $\Delta t < \Delta t_{min}$ **then** $\Delta t = \Delta t_{min}$
07: **for** all $(\tau \in \mathbb{Q}_E)$ {calculate $\Delta E(\tau)$}
08: reorder $\mathbb{Q}_E$ in decreasing order of ΔE
09: **while** $(\mathbb{Q}_E \neq \varnothing)$ {
10: select first task $\tau_{\Delta Emax} \in \mathbb{Q}_E$
11: $t_{\tau_{\Delta Emax}} := t_{\tau_{\Delta Emax}} + \Delta t$
12: update $E_{\tau_{\Delta Emax}}$
13: **for** all $(\tau \in \mathcal{T})$ {update t_S, t_E and t_ϵ}
14: **for** all $(\tau \in \mathbb{Q}_E)$ {**if** $(t_\epsilon(\tau) < \Delta t_{min}) \vee (V_{dd}(\tau) \leq V_t(\tau))$ then $\mathbb{Q}_E := \mathbb{Q}_E - \tau$}
15: $\Delta t = \frac{\min t_\epsilon}{|\mathbb{Q}_E|}$, **if** $\Delta t < \Delta t_{min}$ **then** $\Delta t = \Delta t_{min}$
16: **for** all $(\tau \in \mathbb{Q}_E)$ {update $\Delta E(\tau)$}
17: reorder $\mathbb{Q}_E$ in decreasing order of ΔE
18: }
19: delete MSTG
20: **return** E_Σ, and $V_{dd}(\tau)$ **for** all $(\tau \in \mathcal{T})$

Fig. 5. Pseudo code of the proposed heuristic (PV-DVS) for the generalised DVS problem

scheduled systems the voltage of a single task might not be restricted to one voltage in order to dynamically adapted the system performance to the performance requirements.

3.1.2 *Generalised DVS algorithm for heterogeneous distributed systems.* Having formalised the problem, described the effects of power variations on the voltage selection and the necessity for their consideration in a generalised power model, we introduce next our DVS algorithm. The algorithm, summarised in Figure 5, is based on a constructive heuristic using the defined energy difference (Equation (3)). The starting point of the presented algorithm is a mapped and scheduled task graph (MSTG), i.e., it is known where and in which order the tasks are executed. Execution times and power dissipations are part of the architectural information, which also includes other necessary component properties, like the nominal supply voltage V_{max}, the threshold voltages V_t, etc. The minimal extension time Δt_{min} denotes the minimal time quantum to be distributed in each step of the algorithm. It is defined in order to speed up the determination of the voltage selection by preventing insignificant small extensions leading to trivial power reductions.

To allow for a fast and correct extension of task executions, which might influence other tasks and communications of the system, it is beneficial to capture the schedule and mapping information into the task graph (line 01 in Figure 5). This can be performed by generating a mapped and scheduled task graph, which is a transformed copy of the initial task graph, as shown in Figure 6. The transforma-

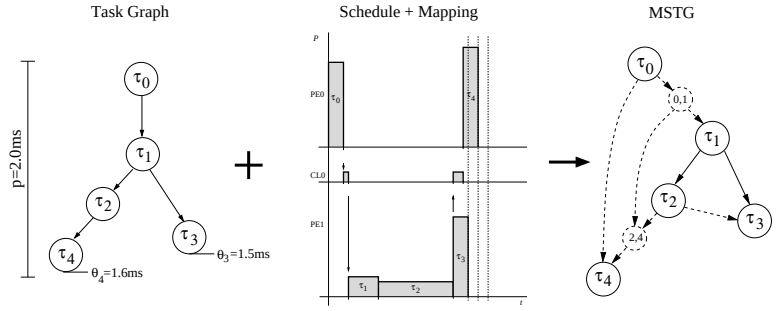


Fig. 6. Capturing the mapping and schedule information into the task graph by using pseudo edges and communication task

tion consists of two steps. Firstly, all communications (edges) that are mapped to communication links are replaced by the so called communication nodes and appropriate edges, thereby preserving the specified functionality. Secondly, all nodes mapped to a certain PE or CL are traversed in chronological order and linked by pseudo-edges [Chretienne et al. 1995], if an edge does not already exist. In this way, the scheduling and mapping are inherited into the task graph and the influence of a task extension can be easily propagated through the system schedule by traversing the MSTG in a breadth-first order to update the start and end times of the activities.

In order to identify all extendable tasks, the algorithm first calculates the available slack times of each hard deadline task (line 03). The algorithm then calculates the slack time t_ϵ of all tasks, taken the interrelation between them into account (line 04). For this purpose an inverse breadth-first search algorithm is used to visit all nodes of the MSTG, in order to inherit the slack time of influenced tasks. If a visited task influences more than a single successor task then the smallest slack of the successor is inherited, in order to guarantee the satisfaction of deadlines. In line 05 the algorithm includes tasks with an available slack time t_ϵ greater or equal than the minimal extension time Δt_{min} into the priority queue $\mathbb{Q}_E$. In this way tasks with negligible small or no extension possibility are excluded from the scaling process. The initial extension time Δt is calculated (line 06 of the algorithm) by dividing the smallest slack time among the extendable tasks, $\min t_\epsilon$, by the number of extendable tasks $|\mathbb{Q}_E|$. This time, however, should not be smaller than the minimal extension time Δt_{min}. It is now possible to calculate the energy difference of all extendable tasks according to Equation (3), as shown in line 07. In line 08, the priority queue $\mathbb{Q}_E$ is reordered in decreasing order of the energy differences.

The algorithm iterates the steps between line 09 and 18 until no extendable tasks are left in the priority queue. In each of these iterations the algorithm picks the first element from the priority queue, the task which leads to the highest energy reduction (line 10). This task is then extended by Δt and the energy dissipation value is updated (lines 11 and 12) according to Equations (1) and (2). In line 13 the extension is propagated through the MSTG, since successor tasks might have been affected by the extension in terms of start time t_S, end time $t_E = t_s + t_{exe}$, and available slack time t_ϵ. In the next step (line 14) inextensible tasks are removed from

the extendable task queue $\mathbb{Q}_E$, if their available slack t_ϵ is smaller than the minimal extension time Δt_{min}, or their scaled supply voltage V_{dd} is small or equal to the threshold voltage V_t. Taking into account the tasks in the new extendable queue, the time quantum Δt is recalculated (line 15) to enable a potential distribution of slack to all tasks in the queue. Based on this Δt value, the energy differences ΔE are updated (line 16). The priority queue $\mathbb{Q}_E$ is reordered according to the new energy differences (line 17). At this point, the algorithm either invokes a new iteration or ends, based on the state of the extendable task queue. If it terminates, the scaling voltages for each task execution and the total dynamic energy dissipation are returned (lines 19 and 20).

The algorithm, as described above, produces scaling voltages under the assumption that variable-voltage PEs are available that support continuous voltage scaling. However, it is possible to adapt the generated scaling voltages towards multi-voltage PEs, which are able to run at a restricted number of predefined voltages. It has been shown in [Ishihara and Yasuura 1998] that the two discrete supply voltages V_{d1} and V_{d2}, $V_{d1} < V_{dd} < V_{d2}$, around the continuous selected voltage V_{dd} are the ones which minimise the energy dissipation, under the assumption that the time overhead for switching between different voltages can be neglected. Thus, our approach can be used for voltage selection on multi-voltage PEs. Given a task τ with execution time t_{exe} at the continuous selected voltage V_{dd}, then, in order to achieve minimal energy consumption, the same task τ will execute on the multiple voltage PE for t_{dis1} time units at the supply voltage V_{dis1} and for t_{dis2} time units at supply voltage V_{dis2}, where

$$t_{exe} = t_{dis1} + t_{dis2} \tag{7}$$

$$t_{dis1} = t_{exe} \cdot \frac{V_{dis1} \cdot (V_{dd} - V_t)^2}{(V_{dis1} - V_t)^2 \cdot V_{dd}} \cdot \frac{\frac{V_{dd}}{(V_{dd}-V_t)^2} - \frac{V_{dis2}}{(V_{dis2}-V_t)^2}}{\frac{V_{dis1}}{(V_{dis1}-V_t)^2} - \frac{V_{dis2}}{(V_{dis2}-V_t)^2}}. \tag{8}$$

Complexity Analysis. The complexity of the proposed PV-DVS algorithm can be calculated as follows: The WHILE loop (line 09) is executed in the worst case $n \cdot m$ times, where $n = |\mathcal{T}|$ is the number of nodes in the graph, since all tasks might be extendable. However, depending on Δt_{min} and Δt, tasks might be extended more than once, and m, for the worst case, is the maximum number of such extensions. The inner part of the WHILE loop shows the following complexities: The propagation of extensions takes $n + c$ in the worst case ($c = |\mathcal{C}|$ is the number of edges in the graph), since all nodes and edges might have to be visited by the breadth-first search (line 13). Removing inextensible tasks, again, might take n steps. Determination of the new extension time Δt is done in most n steps. And finally, updating the extendable queue takes n operations (the queue is implemented as Fibonacci heap). All other calculations inside the WHILE loop are executed in constant time. Therefore, the final time complexity of the proposed PV-DVS algorithm is given as $\mathcal{O}(n \cdot m(4n + c))$. Note that the extendable task queue $\mathbb{Q}_E$ is progressively reduced from length n to zero. The reduction is not uniform since it might occur that suddenly (at the same time) many tasks become inextensible and are excluded from the queue. This, additionally, indicates that the complexity is valid for the worst case. □

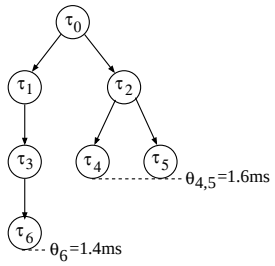


Fig. 7. Second task graph example

3.2 DVS optimised Scheduling

This section is concerned with the scheduling problem for heterogeneous distributed systems containing power managed DVS-PEs. In Section 3.1, we have shown that our generalised DVS algorithm is able to further improve the scaling voltages for the already scheduled tasks, which are mapped to DVS-PEs. However, as mentioned in Section 2, the task scheduling greatly influences how efficiently DVS can be exploited. Simply put, the more slack is available in the schedule, the higher the achieved energy savings by exploiting DVS will be. Again, this becomes more complex and does not hold always for distributed systems under the proposed generalised power model (considering the power profiles) when compared with a fixed power model. In such a case, the available slack for high energy dissipating tasks should be considered more important than the slack of tasks consuming a minor amount of power.

Motivational Example 2: Energy Conscious Scheduling

The purpose of this motivational example is to illustrate the importance to take the PE power profile into account while scheduling tasks and communications in the presence of DVS-PEs. It highlights the importance to take into account the power dissipations for different DVS-PEs, in order to make DVS conscious scheduling decisions.

The specification task graph shown in Figure 7 is mapped to an architecture consisting of three heterogeneous and power managed DVS-PEs, linked through a single bus. Table IV(a) gives the execution time, power dissipation, and the mapping of each task. Additionally, the values for the nominal supply voltage V_{max} and the threshold voltage V_t of each PE are given in Table IV(b). For the sake of simplicity, in this example, the communications are considered to be instantaneous. Figure 8(a) shows a feasible schedule for the mapped tasks, executing at nominal supply voltage. This schedule results in an energy dissipation of $71\mu J$, according to the values given in Table IV(a). It can be observed that task τ_6 has a deadline at $1.4ms$ but it finishes its execution after $1.0ms$, which results in an available deadline slack of $0.4ms$. This slack time can be used to extend the tasks and hence reduce the supply voltage of the PE during the task execution. However, τ_3 and τ_6 are the only extendable tasks, and any other extension of the remaining tasks cannot be tolerated, since task τ_5 finishes execution just on deadline and the tasks τ_0, τ_1, and τ_2 influence the start and end time of task τ_5. Therefore, an optimal DVS schedule

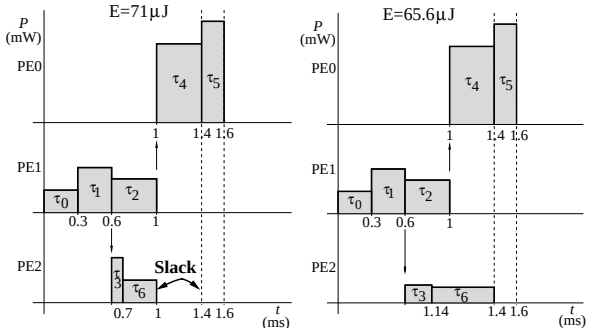
Table III. Task and processing element information

Task	*exec. time (μs)*	*power dis. (mW)*	*mapping*
τ_0	0.30	10	PE1
τ_1	0.30	20	PE1
τ_2	0.40	15	PE1
τ_3	0.10	40	PE0
τ_4	0.40	70	PE2
τ_5	0.20	90	PE2
τ_6	0.30	20	PE0

(a) Execution times, power dissipations, and mapping for the tasks of task graph 2, when running at nominal supply voltage

PE	*Nominal supply voltage V_{max}*	*Threshold Voltage V_t*
0	3.3V	0.8V
1	2.5V	0.6V
2	3.3V	0.8V

(b) Nominal supply voltage and threshold voltage of each DVS-PE

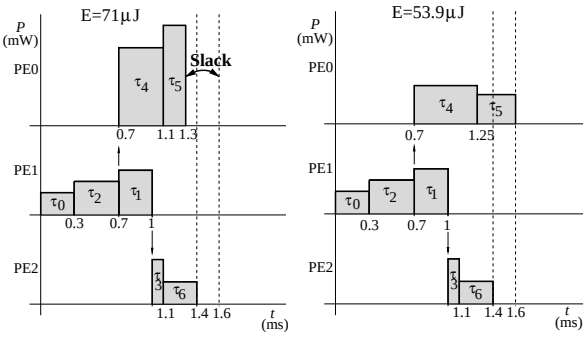


(a) execution at nominal supply voltage

(b) optimal voltage schedule for the given execution order

Fig. 8. A possible schedule, satisfying the given time constraints

for this configuration must only extend tasks τ_3 and τ_6. Taking the power profile of PE2 into account the optimal supply voltages can be calculated as $2.08V$ and $2.34V$ for τ_3 and τ_6, respectively. This results in the reduced power dissipations $P_3 = 6.63mW$ and $P_6 = 5.39mW$. Using the optimised supply voltages, the total



(a) nominal supply voltage schedule

(b) improved voltage schedule executing task τ_4 and τ_5 at $2.74V$ and $2.41V$, respectively.

Fig. 9. Second schedule of the same task graph, satisfying the timing constraints and optimised for PV-DVS

energy dissipation is reduced to $65.61\mu J$, which is a 7.6% reduction.

Now, consider the nominal supply voltage schedule given in Figure 9(a). It can be observed that the energy dissipation is again $71\mu J$, since task mapping and voltages are the same as in Figure 8(b). The given schedule also satisfies all timing constraints. Unlike the previous schedule, in this sequencing of tasks, it is not possible to extend tasks τ_3 and τ_6, since task τ_6 ends its execution just on deadline. However, observing tasks τ_4 and τ_5 reveals an available slack of $0.3ms$. It is important to note that this available slack is smaller than the one given in Figure 8(a), and therefore an energy optimisation neglecting the power variation among PEs and tasks would prefer the solution in Figure 8. Nevertheless, the extensions of task τ_4 and τ_5 allow to scale the PE speed down. The optimal voltages for this configuration can be calculated as $2.74V$ and $2.41V$, for τ_4 and τ_5, respectively. Using this supply voltages allows to reduce the power dissipations to $P_4 = 34.93mW$ and $P_5 = 27.29mW$. In this way the dynamic energy dissipation is reduced to $53.89\mu J$, leading to a 24.1% improvement. This is a significant reduction compared to the achieved 7.6% when the power variations between PEs and tasks are neglected. □

3.2.1 *DVS Optimisation using Genetic List Scheduling.* Our scheduling algorithm for the generalised DVS problem uses a genetic list scheduling approach (GLSA) to optimise the execution order of tasks towards energy reduction and timing feasibility. Unlike previous scheduling approaches that try to improve DVS utilisation using constructive heuristics, the presented algorithm is based on an iterative optimisation which enables a thorough search through the solution space.

It has been shown in [Dhodhi et al. 1995; Grajcar 1999] that the combination of genetic algorithm and list scheduling provides a powerful tool for the synthesis of multiprocessor systems. The main advantages of GLSA approaches over traditional constructive list scheduling methods are:

—The objective, which needs to be optimised, can be based on an arbitrary complex function.

—The enlarged search space (at most $(|\mathcal{T}| + |\mathcal{C}|)!$ different schedules can be produced) provides the opportunity to find solutions of potentially higher quality.

—There is a large freedom to trade-off between acceptable synthesis time and solution quality, as opposed to constructive techniques where only one solution is produced.

—GAs with parallel populations and migration scheme provide a powerful approach to leverage additional computational power of computer clusters, which are becoming more and more commonplace.

—Multi-objective optimisation is an important feature which is supported by genetic algorithms. It provides the opportunity to simultaneously optimise the implementation towards competing goals and allows the system designer to choose among several suitable implementations with different properties.

A detailed functional description of genetic list scheduling approaches can be found in [Dhodhi et al. 1995; Grajcar 1999]. Nevertheless, our implementation varies in two fundamental issues from this previous research:

—Instead of optimising the schedule solely for timing behaviour (reducing the makespan[1]), we consider additionally the issue of energy minimisation with respect to DVS.

—The algorithms described in [Dhodhi et al. 1995] and [Grajcar 1999] employ a list scheduler which determines not only the execution order of tasks but also their mapping. We avoid this combination because of the greediness problems described in [Kalavade 1995] which might lead to infeasible mappings due to exceeded area constraints (memory and gates) of pre-allocated hardware components. A list scheduling, including the mapping step, serially traverses all nodes of the task graphs and maps them to allocated components based on local decisions taken in each step. This might lead to low quality solutions, as opposed to approaches in which mapping is decided in an external loop, based on iterative improvement techniques. Another problem, which occurs when determining the mapping during the list scheduling processes, is that the execution times and power dissipations of the mapped tasks are influenced by the voltage scaling. Therefore, the mapping decisions based on these values might prove to be wrong. For example, mapping a task to a low power consuming ASIC might involve an expensive development of hardware, while the mapping of the same task onto a DVS-enabled ASIP might prove satisfactory when the task execution is scaled.

List scheduling algorithms make scheduling decisions based on task priorities and determine static schedules. Unlike constructive list scheduling techniques that use a sophisticated algorithm for the priority assignment, genetic list scheduling techniques construct and evaluate many different schedules during an iterative priority optimisation process. By encoding the task priorities into a priority string, it becomes possible to utilise genetic operators (crossover and mutation) to change task

[1]Makespan is duration from starting the first task until the last task finishes execution.

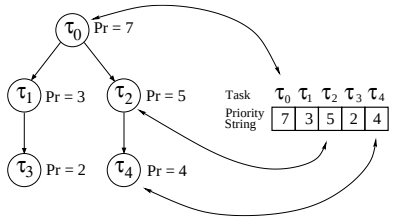


Fig. 10. Task priority encoding into a priority string

priorities and hence generate new scheduling solutions using static list scheduling. Figure 10 shows the encoding and the relations between priority string and tasks. To preserve some string locality, important for an efficient search when using GAs [Goldberg 1989], the priorities are ordered in the same way as visited by a breadth-first search. Now we give an overview of our DVS optimised genetic list scheduling algorithm, as shown in the optimisation Step 2 of Figure 2. The solution pool (25 individuals) of the first generation is initialised half by mobility-based [Wu and Gajski 1990] and half by randomly generated priorities (with values between the lowest and highest mobility), respectively. This initial population was empirically found to be a good starting point, leading to fast convergence. The algorithm then enters the main schedule optimisation loop, which is repeated until no improvement of at least 1% (with respect to the best found feasible schedule) is made for 10 generations. Each iteration of the loop goes successively through the following steps: All new priority candidate strings in the solution pool are used by the list scheduling algorithm to generate schedules at nominal supply voltage. Our implemented list scheduler relies solely on the task priorities to make schedule decisions, i.e., no other techniques, like e.g. hole filling, are used to optimise the schedule. Although such techniques can improve the timing behaviour by eliminating idle periods in the schedule, we dissociate from them since the DVS technique exploits exactly these idle times. The algorithm proceeds by passing the built schedules to the previously presented PV-DVS algorithm (Section 3.1.2), which identifies scaling voltages that minimise the energy dissipation. Note that schedules which exceed hard deadline constraints are still scaled as much as possible and are not excluded from the optimisation, since good solutions are likely to be found as result of transformations performed on invalid configurations. However, a violation penalty is applied in such cases, as explained next. The scaled schedule is evaluated in terms of deadline violations and energy dissipation including DVS reductions. Based on this evaluation, the fitness F_S of each schedule candidate is calculated using the following equation:

$$F_S = \left(\underbrace{\left(\sum_{\tau \in \mathcal{T}} P(\tau) \cdot t_{exe}(\tau) \right)}_{\text{task energy}} + \underbrace{\left(\sum_{\gamma \in \mathcal{C}} P(\gamma) \cdot t_{exe}(\gamma) \right)}_{\text{comm. energy}} \right) \cdot \underbrace{\left(1 + \frac{\sum_{\tau \in \mathcal{T}_d} DV_\tau^2}{T_{HP}^2} \right)}_{\text{Time Penalty}} , \tag{9}$$

$$DV_\tau = \max\big(0, (t_S(\tau) + t_{exe}(\tau)) - t_d(\tau)\big)$$

where P and t_{exe} denote power dissipation and execution time of task τ or communication activity γ, summed to calculate the total dynamic energy dissipation which needs to be minimised. Note that the power dissipations and the execution times of the tasks depend on the found scaling voltages V_{dd}. In order to assign a deadline violation penalty, the energy value is multiplied with a penalty factor based on the sum of the squared deadline violations. T_{HP} is the hyper task graph period (least common multiplier of all task graph periods) used to normalise the deadline violation. Squaring has been applied in order to apply a higher penalty to larger violations of imposed deadlines. By guiding the optimisation with this fitness function, the search for schedules is pushed into regions where low energy and feasible schedules are likely to be found. The algorithm then checks the halting criterion as mentioned above. If the end of the optimisation has not been reached the algorithm continues, and the new priority candidates are ranked and inserted into the solution pool based on their fitness values. Low ranked individuals of the pool are replaced by new ones, which are generated through genetic crossover and mutation. We use a steady state GA, due to its performance advantage compared to generational GAs as indicated in [Rogers and Prügel-Bennett 1999], with a generation gap of 50%, i.e., half of the individuals in the solution pool survive unchanged in each generation. The crossover is carried out by means of a random two point crossover. To avoid a premature convergence towards suboptimal schedules we leverage the idea of a dynamic mutation probability [Fogarty 1989]. This approach gives the algorithm the additional capability to easily escape local minima in the beginning of the optimisation run. The mutation probability follows the equation $1/\exp(N_S \cdot 0.05)$ and is never allowed to drop below 15%. N_S denotes the current generation during the schedule optimisation. At this point, the next iteration is invoked and so different schedules are tried out. The experimental results, given later in Section 4.2, indicate the advantages of our approach in optimising the schedule towards DVS usability when compared to conventional constructive list scheduling approaches.

4. SYNTHESIS EXPERIMENTS

To demonstrate the efficiency and the applicability of the proposed generalised DVS synthesis technique in reducing the energy dissipation of heterogeneous distributed systems containing power managed PEs, we have carried out numerous experiments and comparisons with power neglecting approaches. The PV-DVS and scheduling algorithm as outlined in the previous section have been implemented on a Pentium-III/750MHz Linux PC with 128MB RAM. We have used 68 experimental benchmark examples, partially taken from previously published literature [Gruian 2000; Bambha et al. 2001; Hou and Wolf 1996] and generated using TGFF [Dick et al. 1998], to cover a wide spectrum of application diversity. To demonstrate the real-world applicability of the presented work, we carried out an additional set of experiments on an optical flow detection real-life example. The complexity of the used task graph examples varies between 8 to 100 tasks and 7 to 151 edges. The amount of PEs and CLs in the component libraries varies between 4 and 16. These benchmarks are grouped into five major sets:

(1) Our TGFF generated task graphs (`tgff1-tgff25`) consists of 8 to 100 tasks and are mapped to heterogeneous architectures containing power managed DVS PEs and non-DVS enabled PEs. Therefore, these examples show various power characteristics and component properties. The variations in power are up to 2.6 times on the same PEs. The examples `tgff4_t` and `tgff4_fixed` are identical to `tgff4` with slight modifications; `tgff4_t` denotes a task graph alternative with a critical tight deadline, while `tgff4_fixed` uses only DVS-PEs with a fixed power dissipation.

(2) The examples of Hou et al. [Hou and Wolf 1996] are hypothetical task graphs. `Hou_clustered` represents the same functionality as `Hou`, but the task graph is collapsed from 20 to 8 tasks. Since the initial technology library does not contain any DVS-enabled PEs, we extended the given PEs to DVS-PEs with $V_t = 0.8V$ and $V_{max} = 3.3V$. These examples also show different power dissipations (power variations) among the tasks.

(3) Gruian's and Kuchcinski's graphs [Gruian and Kuchcinski 2001], used in our experiments, represent two sets (`TG1` and `TG2`) of 30 randomly generated communicating tasks with tight deadlines (determined by a critical path scheduling algorithm). These graphs show a high degree of parallelism and are mapped to architectures built of 3 or 10 identical DVS-PEs, assuming constant power consumption. These PEs are multi-voltage processor able to run at $3.3V$, 2.5, $1.7V$, and $0.9V$, while the threshold voltage V_t is $0.4V$.

(4) The applications used by Bambha et al. [Bambha et al. 2001] consist of two differently implemented Fast Fourier Transforms (`fft1` and `fft3`), a Karplus-Strong music synthesis algorithm (`Karp10`), a quadrature mirror filter bank (`qmf4`), and a measurement application (`meas`). These benchmarks are small real-life examples and use architectures composed of 2 to 6 identical DVS-PEs, assuming constant power consumption. Supply voltages are between 0.8 and 7 volts. The throughput constraints and initial average power consumptions are calculated at a reference voltage of 5 volts.

(5) The final benchmarks represents a real-life example, consisting of 32 tasks. It is a traffic monitoring system based on an optical flow detection (OFD) algorithm. This application is a sub-system of an autonomous model helicopter [WITAS ; Gruian and Kuchcinski 2001].

In our experiments, we assume that computation and voltage scaling can be carried out concurrently, as is the case of the processor introduced in [Burd 2001]. Further, we neglected the time overhead needed by the processor to switch between two supply voltages (for real-life DVS processors this is in the range of 10–70μs for a full transition from the highest to the lowest supply voltage and vice versa [Burd 2001]), since the used tasks are considered to be of coarse granularity (in the range of 1–100ms). Therefore, the switching overhead can be considered to be only a small fraction of the total task execution time. However, in the case of fine grained tasks this overhead might influence the voltage selection and should then be considered. All results presented here, except the deterministic ones given in Section 4.1, were obtained by running the optimisation process ten times and averaging the outcomes.

Table IV. Comparison of the presented PV-DVS optimisation with the EVEN-DVS approach (scheduling and mapping are fixed)

Example	*No. of tasks / edges*	**NO-DVS**	**EVEN-DVS**		**Presented Approach**	
		Energy Dissip.	*Energy Dissip.*	*Reduc. (%)*	*Energy Dissip.*	*Reduc. (%)*
tgff1*	8 / 9	355	193.49	45.50	112.87	68.21
tgff2	26 / 43	743224	722412.15	2.80	683954.54	7.97
tgff3	40 / 77	554779	410653.67	25.98	267651.03	51.76
tgff4	20 / 33	431631	402904.08	6.66	375914.03	12.91
tgff4_t	20 / 33	431631	412854.36	4.35	397201.93	7.98
tgff4_fixed	20 / 33	176723	142986.91	19.09	124905.61	29.32
tgff5	40 / 77	4187382	3963647.60	5.34	3767450.25	10.03
tgff6	20 / 26	1419124	1401605.68	1.23	1396445.06	1.60
tgff7	20 / 27	2548751	2289878.34	10.16	1951579.52	23.43
tgff8	18 / 26	1913519	1774151.42	7.28	1668485.33	12.81
tgff9*	16 / 15	996590	974159.01	2.25	918048.34	7.88
tgff10	16 / 21	69352	51263.60	26.08	46483.97	32.97
tgff11	30 / 29	4349627	4293736.56	1.28	4263279.98	1.99
tgff12	36 / 50	2316431	2243710.55	3.14	2212111.25	4.50
tgff13	37 / 36	2912660	2425431.77	16.73	2333338.86	19.89
tgff14	24 / 33	15532	13546.62	12.78	12479.41	19.65
tgff15	40 / 63	62607	62078.93	0.84	60334.62	3.63
tgff16	31 / 56	3494478	2913341.14	16.63	2518711.99	27.92
tgff17	29 / 56	23459	20396.41	13.06	18334.01	21.85
tgff18	12 / 15	1851688	1851687.99	0.00	1526059.97	17.59
tgff19	14 / 19	5939	4713.59	20.63	4395.37	25.99
tgff20*	19 / 25	77673	48334.30	37.77	40280.98	48.14
tgff21	70 / 99	3177705	3175497.22	0.07	2658534.22	16.34
tgff22	100 / 135	5821498	5036657.40	13.48	4445545.63	23.64
tgff23*	84 / 151	11567283	10791880.89	6.70	10133912.03	12.39
tgff24	80 / 112	5352217	5349024.86	0.06	5238478.58	2.13
tgff25	49 / 92	5735038	5648816.00	1.50	5502681.64	4.05
Hou*	20 / 29	13712	10337.05	24.61	7474.55	45.49
Hou_clust.*	8 / 7	14546	11543.35	20.64	10270.32	29.39

*Components used for these examples consists of DVS-PEs only

4.1 Performance of the Generalised DVS Algorithm

To demonstrate the influence of power variations on the efficiency of DVS, we compare our approach, which takes the power profile into account, with a power neglecting approach. This power neglecting approach (in the following referred to as EVEN-DVS) is based on the idea to distribute available slack time *evenly* among the processing elements, somewhat similar to the voltage scaling idea used in [Luo and Jha 2000]. However, since the mapping and scheduling approach proposed in [Luo and Jha 2000] targets also additional different objectives, a direct comparison is not valid.

Table IV shows a comparison between the EVEN-DVS and the proposed DVS approach. In order to judge the complexity of the individual benchmark examples, the table gives the number of nodes and edges in the task graphs. The comparison between the two DVS approaches is carried out with respect to the energy dissipation when no DVS is employed (see Column NO-DVS). Consider for example benchmark `tgff17`, which consist of 29 tasks and 56 communications between tasks. The unscaled execution (NO-DVS) of the application dissipates an energy of 23459. Using an even distribution of slack time (EVEN-DVS) this power consumption can be reduced to 20396, a reduction of 13.1%. However, using the proposed generalised DVS algorithm the dissipated energy is further reduced to 18334, when

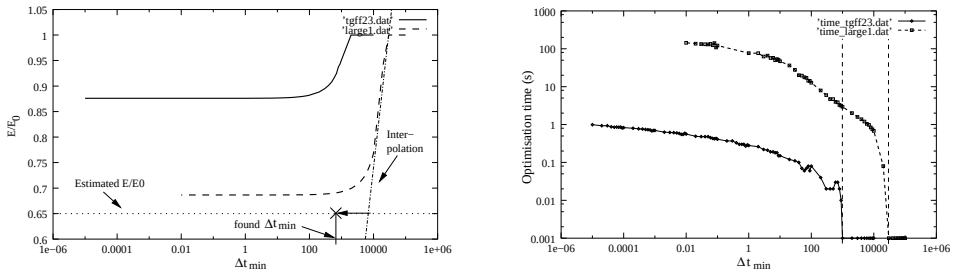
Table V. PV-DVS results using the benchmark set of Bambha et al.

Example	*No. of Nodes/ Edges*	**NO-DVS**	**Proposed Approach**		
		Energy Dissip.	*Energy Dissip.*	*CPU time (s)*	*Reduction (%)*
fft1	28/32	29600	18172	0.21	38.61
fft3	28/32	48000	36890	0.14	23.15
karp10	21/20	59400	44038	0.12	25.86
meas	12/12	28300	25973	0.11	8.22
qmf4	14/21	16000	12762	0.11	20.24

compared to NO-DVS a reduction of 21.8%.

For all examples shown in Table IV it is assumed that the mapping and scheduling have been pre-determined, using a fixed mapping and a schedule generated by a mobility based list scheduling. Thus, the energy reductions are solely achieved through voltage scaling. As expected, both the EVEN-DVS and the presented scaling technique reduced the energy dissipation of the systems in all cases (Column 6 and 9), except for `tgff18` where the even distribution of slack could not achieve any improvement. It can be observed that the proposed DVS heuristic was able to further improve the energy dissipation of all examples, when compared to EVEN-DVS. Even in the case of `tgff18` a reduction of 17.8% could be achieved. Due to our particular implementation of the DVS algorithm which distributes slack evenly among the PEs (EVEN-DVS), also slack is allocated on non-DVS-PEs. Therefore, the higher energy reduction of the proposed DVS algorithm are due to two facts. Firstly, EVEN-DVS allocates slack time on non-DVS-PEs. These times, of course, cannot be exploited to lower the power consumption. Secondly, the proposed DVS technique considers the power profile information during the voltage scaling. This leads to better energy reductions (see Motivational Example 1). To distinguish between both effects, we have indicated in Table IV the architectures which consists of DVS-PEs only. In these examples, the higher energy reduction in solely achieved by taken the power profile into account. The remaining examples achieve the increased energy efficiency due to both effects. We have further conducted experiments with the benchmark set used by Bambha et al. [Bambha et al. 2001]. Since they use a different communication model (contention, requests for the bus, etc.), we had to re-calculate the throughput constraints. Therefore, a direct comparison between the results reported in [Bambha et al. 2001] and the results presented here is not possible. Nevertheless, the re-calculation of the throughput was carried out for the same task mapping and execution order as in [Bambha et al. 2001], which is based on a dynamic level scheduling approach [Sih and Lee 1993]. The results of these five examples, scaled by our PV-DVS method, are given in Table V. It can be observed that in all cases the energy was reduced by 8.22 to 38.61%. Further, the highly serialised structure of `meas` allowed us to calculate the theoretically optimal voltage schedule for this example. Using this optimal supply voltages results in 13% energy saving. Our PV-DVS algorithm achieved for this example a reduction of 8.22%, which is only 4.78% higher than the theoretically optimal solution.

To give insight into the dependencies between the computational efforts, solution quality, and the minimum extension time Δt_{min} (see also the complexity analysis in Section 3.1.2), we have conducted two experiments. In order to achieve accurate results, especially for the time measurement, the experiments are carried out using



(a) Energy reduction quality dependent on minimum extension time Δt_{min}

(b) Execution time dependent on minimum extension time Δt_{min}

Fig. 11. The influence of Δt_{min} on energy reduction and optimisation time

two large task graphs with 80 (`tgff23`) and 400 (`large1`) tasks. Figure 11(a) illustrates the dependency between the minimum extension time Δt_{min} and the solution quality (given as reduced energy E over nominal energy E_0). It can be observed that no energy reduction can be achieved until the Δt_{min} is smaller than the largest slack available in the task schedule (2364 for `tgff23` and 29581 for `large1`, see Figure 11(a)). Certainly, if the algorithm must distribute time quanta bigger than any slack, it cannot perform any voltage reduction, and therefore $E/E_0 = 1$. At the same time, it is not desirable to decrease the minimum extension time too much since the additional reductions become insignificant (the curves level out) and will only increase the computational time of the optimisation. Figure 11(b), which gives the dependency between minimum extension time Δt_{min} and execution time of the DVS algorithm. It is therefore important to find a good value for Δt_{min}, which trades-off between solution quality and optimisation time. In our experiments we use the following heuristic approach to find an appropriate Δt_{min} setting for each solution candidate. It is based on the observation that for all used benchmarks the characteristics shown in Figure 11(a) and Figure 11(b) hold.

With reference to Figure 11(a), we interpolate the nearly linear (in a semi-logarithmic scale) energy drop, after decreasing Δt_{min} below the highest DVS slack, using the logarithmic function,

$$y = \alpha \cdot \log x + \beta \tag{10}$$

where the constants α and β are calculated using two initial points in the quasi linear part of the graph. The first point corresponds to the highest available slack s_h on any of the DVS-PEs, hence, it matches the nominal energy dissipation. This point can be found in linear time. To establish the second point, needed for the interpolation, the DVS algorithm is run with an Δt^*_{min} three times smaller than the highest DVS slack to find its corresponding reduced energy dissipation E^*. For all used examples this was still in the steeply dropping part of the graph. Using these points, the constants α and β are given by

$$\alpha = \frac{1 - E^*}{\log(s_h) - \log(\Delta t^*_{min})} \qquad \beta = E^* - \alpha \cdot \log(\Delta t^*_{min})$$

Such a linear interpolation is shown in Figure 11(a) for the `large1` example. Of course, finding the second point has a computational overhead, however, as it can be seen from Figure 11(a) and Fig 11(b), this "investment" pays off when compared to a wrong choice of Δt_{min}, which could results in a much higher computational time or a much higher energy consumption. The next step towards a good value for Δt_{min} is it to find a "rough" estimation for the achievable energy reduction. We calculate the estimation for the scaled energy consumption based on the average power dissipation on each DVS-PE and the sum of the maximal available slack on these PEs. An estimated energy dissipation for `large1` is indicated in Figure 11(a). The minimum extension time Δt_{min} could be set to the intersection of the energy estimation and the interpolated energy drop (as show in Figure 11(a)). However, we set it one order of magnitude lower, as indicated by an arrow in the figure. This is done to account for the fact that in the case of an energy estimation close to the real achievable energy reduction, the intersection would be approximately one order of magnitude to high. In the case that the energy estimation would be far below the real achievable energy reduction, the calculated Δt_{min} would become unnecessary small. Therefore, we allow no Δt_{min} smaller than 2.5 orders of magnitude compared to the maximal DVS slack. This is based on the observation that all used benchmarks show a similar characteristic, and ideal Δt_{min} can be found at maximal 2.5 orders of magnitude from the maximal DVS slack.

4.2 Schedule optimisation using the Generalised DVS Approach

To assess the capability of the proposed DVS optimised genetic list scheduler (presented in Section 3.2) to reduced the power consumption as well as finding feasible schedules, we have conducted several experiments. Table VI shows, for the same benchmarks as in the previous section, the achieved energy reductions and computational overheads, after including the EVEN-DVS and our PV-DVS algorithm inside the schedule optimisation loop. Comparing the achieved reductions (Table IV) with the results obtained by a mobility based scheduling (Table VI) reveals that for most examples the energy consumption was reduced, i.e., the schedule optimisation was able to find execution orders which allow a more effective exploitation of DVS. For instance, consider benchmark `tgff23`. We can observe that the energy reduction was increase from 6.7% to 15.05% when using EVEN-DVS and from 12.39% to 23.44% when utilising PV-DVS. Certainly, the GA based schedule optimisation introduces a computational overhead which results in a necessary trade-off between energy reduction and optimisation time. Of course, the linear time complexity of the EVEN-DVS approaches results in lower optimisation times compared to PV-DVS which has a polynomial complexity. However, the achieved reductions justify this overhead, which is in the worst case $21.2s$ compared to $0.59s$ for a task graph with 84 nodes (`tgff23`).

To further confirm the quality of the proposed DVS optimised scheduling technique, we compare it next with the DVS scheduling approach proposed by Gruian et al. [Gruian and Kuchcinski 2001], using the benchmark collections `TG1` and `TG2`, which contain 60 task graph examples. The reported average energy reductions in [Gruian and Kuchcinski 2001] are 28% and 13% for the tight deadline task graph collections `TG1` and `TG2`, respectively. Table VII presents the results obtained using the proposed DVS optimised scheduling technique. The table is divided into

Table VI. Experimental results obtained using the generalised DVS algorithm integrated into a genetic list scheduling heuristic

Example	EVEN-DVS + GLSA			PV-DVS + GLSA		
	Energy	CPU time (s)	Reduction (%)	Energy	CPU time (s)	Reduction (%)
Tgff1	191	0.12	46.27	102	0.14	71.16
Tgff2	572920	0.20	22.91	545451	0.27	26.61
Tgff3	266907	0.33	51.89	170838	3.11	69.21
Tgff4	377445	0.24	12.55	375778	0.97	12.94
Tgff4_t	405473	0.25	6.06	396579	0.62	8.12
Tgff4_fixed	127867	0.26	27.65	124419	1.00	29.60
Tgff5	3721137	0.37	11.13	3450292	2.41	17.60
Tgff6	1399968	0.23	1.35	1396445	0.25	1.60
Tgff7	1925000	0.20	24.47	1797520	0.27	29.47
Tgff8	1722056	0.19	10.01	1648322	0.20	13.86
Tgff9	829608	0.18	16.76	774994	0.26	22.24
Tgff10	45325	0.17	34.65	44529	0.22	35.79
Tgff11	3755206	0.22	13.67	3621740	0.42	16.73
Tgff12	2212405	0.34	4.49	2198978	3.73	5.07
Tgff13	2342892	0.28	19.56	2315766	0.80	20.49
Tgff14	11891	0.21	23.44	11753	0.25	24.33
Tgff15	61271	0.41	2.13	60129	1.07	3.96
Tgff16	2492365	0.26	28.68	2449747	0.55	29.90
Tgff17	18923	0.27	19.34	18249	0.56	22.21
Tgff18	1724421	0.14	6.87	1421224	0.16	23.25
Tgff19	4515	0.17	23.98	4357	0.16	26.63
Tgff20	42704	0.19	45.02	37223	0.60	52.08
Tgff21	2983044	0.56	6.13	2578046	3.92	18.87
Tgff22	4664876	1.19	19.87	4119749	3.44	29.23
Tgff23	9826644	0.64	15.05	8855575	21.25	23.44
Tgff24	5240977	0.59	2.08	4881188	10.48	8.80
Tgff25	4922085	0.39	14.18	4545250	1.91	20.75
Hou	10211	0.21	25.53	7474	0.23	45.49
Hou_clustered	11543	0.18	20.64	10270	0.12	23.39

Table VII. Experimental results obtained using our generalised DVS optimised scheduling approach for benchmark examples `TG1` and `TG2`

Benchmark	Continuous Reduction (%)	Discrete Reduction (%)	CPU time (s)
TG1	41.52	37.86	3.96
TG2	18.90	15.93	0.74

the two benchmark collections. Although the examples do not allow our approach to leverage power variations, since the specified power values are constant, the achieved energy reduction for `TG1` and `TG2` are 41.52% and 18.90% (Column 5 and 11), respectively. This is an improvement of 13.52% and 5.90%, which indicates the effectiveness of the proposed optimisation technique, even when using constant power benchmark examples. However, since the results in [Gruian and Kuchcinski 2001] are obtained using multi-voltage PEs rather than variable-voltage PEs, we have conducted an additional set of experiments, using the same multiple voltages as given in [Gruian and Kuchcinski 2001]. Each supply voltage found by our PV-DVS algorithm is split into its two neighbouring discrete voltages of the multi-voltage PE, and the corresponding run-times for each voltage are calculated

Table VIII. Mapping optimisation of the benchmark set TG1 using NO-DVS, EVEN-DVS, and PV-DVS

Example	**NO-DVS**		**EVEN-DVS + MOB**			**PV-DVS + GLSA**			
	Energy Dissip.	*time (s)*	*Energy Dissip.*	*time (s)*	*Red. (%)*	*Energy Dissip.*	*time (s)*	*Red. (%)*	*Red. Fac.*
r000	798700	53.87	unsolved	18.60	n/a	586806	194.86	26.53	n/a
r001	759500	56.16	592674	13.87	21.97	399839	804.73	47.35	2.16
r002	744800	55.64	unsolved	16.51	n/a	551944	189.97	25.89	n/a
r003	994700	27.76	711887	15.98	28.43	554171	769.58	44.29	1.56
r004	886900	54.00	unsolved	19.97	n/a	566263	360.58	36.15	n/a
r005	744800	54.94	465853	16.75	37.45	373677	1596.67	49.83	1.33
r006	901600	36.88	unsolved	17.55	n/a	589469	827.22	34.62	n/a
r007	837900	55.20	unsolved	20.20	n/a	565731	269.07	32.48	n/a
r008	862400	30.63	unsolved	19.25	n/a	635426	207.46	26.32	n/a
r009	681100	53.24	424723	14.99	37.64	311751	1535.28	54.23	1.44

using Equations (7) and (8). The results of the discrete voltage optimisation are shown in Table VII (see columns with the headings "Discrete Reduc."). For the two benchmark sets the achieved average energy reductions are 37.86% and 15.93%, respectively, which represent improvements of 9.86% and 2.93%. Note that these reductions were obtained on benchmarks which do *not* show any power variations and so this optimisation feature of the proposed DVS algorithm stays unexploited. The achieved improvements are due to the fact that our iterative GA-based approach is able to explore a large space of potentially energy saving schedules, as opposed to the constructive list scheduling used in [Gruian and Kuchcinski 2001]. Regarding the computational times, Gruian et al. reported average times for the 30-node task graphs of $10s$ to $120s$, while the proposed algorithm executes on average in $0.74s$ to $3.96s$, indicating a performance advantage of the presented scaling technique.

Another feature of the proposed scheduling approach is important to be mentioned. The scheduling optimisation (GLSA) does not only reduce significantly the dissipated energy in the presents of DVS-PEs, but also increases the possibility to find feasible schedules, when compared to constructive techniques, such as mobility based scheduling. This is of great importance since high quality solutions could be found in design space regions where infeasible and feasible solutions are spatially placed closely together. Making a wrong decision might involve a more costly implementation of the system specification. To clarify this, consider the results obtained with the benchmark set TG1 from Gruian et al. [Gruian and Kuchcinski 2001], as shown in Table VIII. The results shown in Column 4 are based on EVEN-DVS and a constructive list scheduling heuristic which uses the mobility of tasks as priorities. Consider for example benchmark r000. In the case of this benchmark the scheduling attempt fails and the implementation is infeasible (Column 4, unsolved), making it necessary to increase the performance of the allocated system for the given mapping. On the other hand, our iterative GA-based list scheduling technique (GLSA) is able to improve infeasible schedules by providing feedback to the optimisation process and therefore feasible schedules might be found, as in the case of the task graph example r000 (Column 7). This effect is likely to appear in the presence of tight deadline specifications, as it is the case with the benchmark set TG1. It can be observed that for 6 out of 10 examples *no* feasible mapping could be found when using a mobility based scheduling algorithm. Similarly, for

Table IX. Increasing architectural parallelism to allow voltage scaling of the OFD algorithm

Architecture	Static Power (W)	Dynamic Power (W)	Total Power (W)	Reduction (%)	CPU time (s)
2 DSPs	0.383	2.137	2.52	–	–
3 DVS-DSPs	0.574	1.563	2.137	15.2	0.49
4 DVS-DSPs	0.736	1.053	1.789	29.0	0.69
5 DVS-DSPs	0.898	1.000	1.898	24.7	0.76

the remaining 20 task graphs of the `TG1` benchmark set only 8 could be scheduled using a mobility based scheduling approach. Clearly, the improved schedules are solely introduced by the GA-based list scheduling and are not dependent on the different voltage scaling approaches.

In addition to the experiments presented above, we have validated the energy reduction capability of the proposed scheduling and voltage scaling techniques, using the real-life example of an optical flow detection (OFD) algorithm. This application is part of an autonomous helicopter and used for traffic monitoring purpose. In its current implementation the OFD algorithm runs on two ADSP-21061L digital signal processors (DSPs), with an average current of $760mA$ at $3.3V$, resulting in an average power dissipation of approximately $2.5W$. However, due to the stringent power budget on board of the helicopter, including application critical sub-systems, it is necessary to keep the overall power dissipation under a certain limit. With respect to the performance of the two DSPs, this implementation is able to process 12.5 frames of 78x120 pixels per second. We have conducted two set of experiments regarding the OFD algorithm. In both we consider an hypothetical extension of the DSPs towards DVS capability (DVS-DSP) and take into account that such an extension increases the static power consumption of the processors. This was estimated to be 10% for the systems presented in [Pering et al. 1998].

In the first experiment the performance constraint is kept fixed, i.e., the flow detection has to perform 12.5 frames per second. Since the 2 DSP implementation needs to utilise the processors completely to achieve the $12.5Hz$ repetitions, we increase the system performance by allocating additional DSPs. In this way it is possible to utilise the application parallelism more effectively and hence achieve a high performance. This over performance can then be exploited by the DVS-DSPs, in order to lower the dynamic power consumption. Table IX reports on our findings. From this table it can be observed that with increasing number of PEs the static power consumption increases as well, while the dynamic power consumption decreases. Nevertheless, from the battery point of view the total power dissipation is the limiting factor and it can be seen that the implementation with 4 DVS-DSPs shows the lowest power consumption. It is important to note that the implementations shown in Table IX do not necessitate any performance degradation, though the energy dissipation is reduced by up to 29%. The proposed scheduling and voltage scaling techniques optimised the execution of the 32 tasks in less than $0.8s$. Of course, the more DVS-DSPs are allocated, the more costly the implementation becomes.

The last experiment is based on the fact at a $12.5Hz$ repetition rate is unnecessary high. We therefore relax the performance constraints to a repetition rate of $8.33Hz$, which is still high enough to allow a correct flow detection, i.e., a cor-

Table X. Relaxed performance constraints of the OFD algorithm at $8.33Hz$

Architecture	Static Power (W)	Dynamic Power (W)	Total Power (W)	Reduction (%)	CPU time (s)
2 DSPs	0.383	2.137	2.52	–	–
2 DVS-DSPs	0.413	0.766	1.179	53.2	1.10
3 DVS-DSPs	0.574	0.699	1.273	49.5	1.78
4 DVS-DSPs	0.736	0.497	1.233	51.1	2.27
5 DVS-DSPs	0.898	0.503	1.401	44.4	3.54

rect operation of the OFD algorithm. In this case even the implementation build out of 2 PEs is not fully utilised and the resulting idle times can be exploited by DVS to reduced the power consumption. Table X shows the results for different architectural alternatives, consisting of 2 to 5 DVS-DSPs. Among all alternatives, the system built out of two DVS-PEs is the clear favourite, since it achieves the lowest energy consumption at the lowest cost. Clearly, the dynamic power reductions achieved for the 3–5 DVS-DSP systems do not justify the increased the static power consumption. The optimisation of schedule and voltage scaling for these system were carried out in at most $3.54s$.

5. CONCLUSIONS

In this work, we have demonstrated that the consideration of power variations is essential during the energy optimised synthesis of heterogeneous distributed hardware/software systems containing power managed PEs, especially in the presence of DVS-PEs. This has been mostly neglected in previous work on distributed systems which include DVS-PEs. We have presented a novel DVS algorithm which identifies supply voltages for the tasks executing on DVS-PEs, under the consideration of power variation effects in order to minimise the dynamic energy dissipation. The approach is based on the defined energy difference. This DVS technique was successfully integrated into a genetic list scheduling approach as to iteratively optimise a mapped system specification towards an efficient exploitation of the available DVS-PEs. The integration was achieved by adapting the employed list scheduler for the particular problems involved in dynamic voltage scaling. We have further comprehensively investigated the effects of scheduling on the achievable energy reductions when the generalised DVS technique is employed. The extensive experimental results show the necessity to take the PE power profiles into account when optimising the scaled supply voltages for energy minimisation. Due to recent developments in embedded computing systems and the availability of various implementations of state-of-the-art DVS processors [Intel® XScale™ 2000; Mobile AMD Athlon™4 2000; Klaiber 2000] with power management techniques, voltage scaling algorithms (as the present one) are becoming an important part of the synthesis flow.

Acknowledgements

The authors wish to thank EPSRC for the financial support in this project. They would also like to thank Flavius Gruian (Lund University, Sweden) and Neal K. Bambha (University of Maryland, USA) for kindly providing their benchmark sets. Additionally, the authors wish to thank the reviewers for their excellent critical assessment of the work and their useful suggestions.

REFERENCES

Bambha, N., Bhattacharyya, S., Teich, J., and Zitzler, E. 2001. Hybrid Global/Local Search Strategies for Dynamic Voltage Scaling in Embedded Multiprocessors. In *Proc. 1st Int. Symp. Hardware/Software Co-Design (CODES'01)*. 243–248.

Brandolese, C., Fornaciari, W., Salice, F., and Sciuto, D. 2000. Energy Estimation for 32 bit Microprocessors. In *Proc. 8th Int. Workshop Hardware/Software Co-Design (CODES'00)*. 24–28.

Burd, T. D. 2001. Energy-Efficient Processor System Design. Ph.D. thesis, University of California at Berkeley.

Burd, T. D. and Brodersen, R. W. 1996. Processor Design for Portable Systems. *J. VLSI Signal Processing 13,* 2 (August), 203–222.

Burd, T. D., Pering, T. A., Stratakos, A. J., and Brodersen, R. W. 2000. A Dynamic Voltage Scaled Microprocessor System. *IEEE J. Solid-State Circuits 35,* 11 (November), 1571–1580.

Chretienne, P., Coffman, E. G., Lenstra, J. K., and Liu, Z. 1995. *Scheduling Theory and its Applications*. John Wiley & Sons.

Devadas, S. and Malik, S. 1995. A Survey of Optimization Techniques Targeting Low Power VLSI Circuits. In *Proc. IEEE 32nd Design Automation Conf. (DAC95)*. 242–247.

Dhodhi, M. K., Ahmad, I., and Storer, R. 1995. SHEMUS: Synthesis of Heterogeneous Multiprocessor Systems. *J. Microprocessors and Microsystems 19,* 6 (August), 311–319.

Dick, R., Rhodes, D., and Wolf, W. 1998. TGFF: Task Graphs for free. In *Proc. 5th Int. Workshop Hardware/Software Co-Design (Codes/CASHE'97)*. 97–101.

Dick, R. P. and Jha, N. K. 1998. MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Co-Synthesis of Distributed Embedded Systems. *IEEE Trans. Computer-Aided Design 17,* 10 (Oct), 920–935.

Eles, P., Peng, Z., Kuchcinski, K., and Doboli, A. 1997. System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search. *J. Design Automation for Embedded Systems 2*, 5–32.

Ernst, R., Henkel, J., and Brenner, T. 1993. Hardware-Software Co-synthesis for Mirco-Controllers. *IEEE Design & Test of Comp. 10,* 4 (Dec), 64–75.

Fogarty, T. C. 1989. Varying the probability of mutation in the genetic algorithm. In *Proc. 3rd Int. Conf. Genetic Algorithms (ICGA)*. 104–109.

Fornaciari, W., Sciuto, D., and Silvano, C. 1999. Power Estimation for Architectural Exploration of HW/SW Communication on System-Level Buses. In *Proc. 7th Int. Workshop Hardware/Software Co-Design (CODES'99)*. 152–156.

Garey, M. R. and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the theory of NP-Completeness*. W.H. Freeman and Company.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley Publishing Company.

Grajcar, M. 1999. Genetic List Scheduling Algorithm for Scheduling and Allocation on a Loosely Coupled Heterogeneous Multiprocessor System. In *Proc. IEEE 36th Design Automation Conf. (DAC99)*. 280–285.

Gruian, F. 2000. System-Level Design Methods for Low-Energy Architectures Containing Variable Voltage Processors. In *Workshop Power-Aware Computing Systems*.

Gruian, F. and Kuchcinski, K. 2001. LEneS: Task Scheduling for Low-Energy Systems Using Variable Supply Voltage Processors. In *Proc. Asia South Pacific - Design Automation Conf. (ASP-DAC'01)*. 449–455.

Gutnik, V. and Chandrakasan, A. 1997. Embedded Power Supply for Low-Power DSP. *IEEE Trans. VLSI Systems 5,* 4 (425–435).

Henkel, J., Benner, T., and Ernst, R. 1993. Hardware Generation and Partitioning Effects in the COSYMA System. In *Proc. Int. Workshop Hardware/Software Co-Design (Codes/CASHE'93)*.

Henkel, J. and Ernst, R. 2001. An Approach to Automated Hardware/Software Partitioning using a Flexible Granularity that is driven by High-Level Estimation Techniques. *IEEE Trans. VLSI Systems 9,* 2, 273–289.

Hong, I., Kirovski, D., Qu, G., Potkonjak, M., and Srivastava, M. B. 1999. Power Optimization of Variable-Voltage Core-Based Systems. *IEEE Trans. Computer-Aided Design 18,* 12 (Dec), 1702–1714.

Hou, J. and Wolf, W. 1996. Process Partitioning for Distributed Embedded Systems. In *Proc. CODES*. 70 – 76.

Intel® XScale™. 2000. Developer's Manual. Order Number 273473-001.

Ishihara, T. and Yasuura, H. 1998. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In *Proc. Int. Symp. Low Power Electronics and Design (ISLPED'98)*. 197–202.

Kalavade, A. 1995. System-Level Codesign of Mixed Hardware-Software Systems. Ph.D. thesis, University of California, Berkeley.

Kirovski, D. and Potkonjak, M. 1997. System-level Synthesis of Low-Power Hard Real-Time Systems. In *Proc. IEEE 34th Design Automation Conf. (DAC97)*. 697–702.

Klaiber, A. 2000. The Technology behind Crusoe Processors. http://www.transmeta.com.

Lee, S. and Sakurai, T. 2000. Run-time Voltage Hopping for Low-power Real-time Systems. In *Proc. IEEE 37th Design Automation Conf. (DAC00)*. 806–809.

Li, Y.-T. S., Malik, S., and Wolfe, A. 1995. Performance Estimation of Embedded Software with Instruction Cache Modeling. In *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD-95)*. 380–387.

Liu, J., Chou, P. H., Bagherzadeh, N., and Kurdahi, F. 2001. Power-Aware Scheduling under Timing Constraints for Mission-Critical Embedded Systems. In *Proc. IEEE 38th Design Automation Conf. (DAC01)*. 840–845.

Luo, J. and Jha, N. K. 2000. Power-conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-time Embedded Systems. In *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD-00)*. 357–364.

Luo, J. and Jha, N. K. 2001. Battery-aware Static Scheduling for Distributed Real-Time Embedded Systems. In *Proc. IEEE 38th Design Automation Conf. (DAC01)*. 444–449.

Manzak, A. and Chakrabarti, C. 2000. Variable Voltage Task Scheduling for Minimizing Energy or Minimizing Power. In *Proc. Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP00)*. 3239–3242.

Micheli, G. D. and Gupta, R. K. 1997. Hardware/Software Co-Design. In *Proceedings of the IEEE*. 349–365.

Mobile AMD Athlon™4. 2000. Processor Model 6 CPGA Data Sheet. Publication No 24319 Rev E.

Muresan, R. and Gebotys, C. H. 2001. Current Consumption Dynamics at Instruction and Program Level for a VLIW DSP Processor. In *Proc. Int. Symp. System Synthesis (ISSS'01)*. 130–135.

Okuma, T., Ishihara, T., and Yasuura, H. 1999. Real-Time Task Scheduling for a Variable Voltage Processor. In *Proc. Int. Symp. System Synthesis (ISSS'99)*. 24–29.

Okuma, T., Ishihara, T., and Yasuura, H. 2001. Software Energy Reduction Techniques for Variable-Voltage Processors. *IEEE Design & Test of Comp. 18,* 2 (March–April), 31–41.

Pedram, M. 1996. Power Minimization in IC Design: Principles and Applications. *ACM Trans. Design Automation of Electronic Systems (TODAES) 1,* 1 (Jan), 3–56.

Pering, T., Burd, T. D., and Brodersen, R. B. 1998. The Simulation and Evaluation for Dynamic Voltage Scaling Algorithms. In *Proc. Int. Symp. Low Power Electronics and Design (ISLPED'98)*. 76–81.

Prakash, S. and Parker, A. 1992. SOS: Synthesis of Application-Specific Heterogeneous Multiprocessor Systems. *J. Parallel & Distributed Computing*, 338–351.

Quan, G. and Hu, X. S. 2001. Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors. In *Proc. IEEE 38th Design Automation Conf. (DAC01)*. 828–833.

Quan, G. and Hu, X. S. 2002. Minimum Energy Fixed-Priority Scheduling for Variable Voltage Processors. In *Proc. Design, Automation and Test in Europe Conf. (DATE2002)*. 782–787.

Rogers, A. and Prügel-Bennett, A. 1999. Modelling the dynamics of a steady-state genetic algorithm. In *Foundations of Genetic Algorithms (FOGA-5)*. 57–68.

Schmitz, M. T. 2003. Energy Minimisation Techniques for Distributed Embedded Systems. Ph.D. thesis, University of Southampton.

Schmitz, M. T. and Al-Hashimi, B. M. 2001. Considering Power Variations of DVS Processing Elements for Energy Minimisation in Distributed Systems. In *Proc. Int. Symp. System Synthesis (ISSS'01)*. 250–255.

Schmitz, M. T., Al-Hashimi, B. M., and Eles, P. 2002. Energy-Efficient Mapping and Scheduling for DVS Enabled Distributed Embedded Systems. In *Proc. Design, Automation and Test in Europe Conf. (DATE2002)*. 514–521.

Shin, Y. and Choi, K. 1999. Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems. In *Proc. IEEE 36th Design Automation Conf. (DAC99)*. 134–139.

Shin, Y., Choi, K., and Sakurai, T. 2000. Power Optimization of Real-Time Embedded Systems on Variable Speed Processors. In *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD-00)*. 365–368.

Sih, G. C. and Lee, E. A. 1993. A Compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parallel and Distributed Systems 4,* 2 (Feb.), 175–187.

Simunic, T., Benini, L., Acquaviva, A., Glynn, P., and Micheli, G. D. 2001. Dynamic Voltage Scaling and Power Management for Portable Systems. In *Proc. IEEE 38th Design Automation Conf. (DAC01)*. 524–529.

Teich, J., Blickle, T., and Thiele, L. 1997. An Evolutionary Approach to System-Level Synthesis. In *Proc. 5th Int. Workshop Hardware/Software Co-Design (Codes/CASHE'97)*. 167 – 171.

Tiwari, V., Malik, S., and Wolfe, A. 1994. Power Analysis of Embedded Software: A First Step Towards Software Power Minimization. *IEEE Trans. VLSI Systems*.

Weiser, M., Welch, B., Demers, A., and Shenker, S. 1994. Scheduling for Reduced CPU Energy. In *Proc. USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 13–23.

WITAS. The Wallenberg laboratory for research on Information Technology and Autonomous System. http://www.ida.liu.se/ext/witas/.

Wolf, W. H. 1994. Hardware/Software Co-Design of Embedded Systems. In *Proceedings of the IEEE*. 967–989.

Wolf, W. H. 1997. An Architectural Co-Synthesis Algorithm for Distributed, Embedded Computing Systems. *IEEE Trans. VLSI Systems 5,* 2 (June), 218–229.

Wu, M. and Gajski, D. 1990. Hypertool: A Programming Aid for Message-passing Systems. *IEEE Trans. Parallel and Distributed Systems 1,* 3 (July), 330–343.

Xie, Y. and Wolf, W. 2001. Allocation and Scheduling of Conditional Task Graph in Hardware/Software Co-Synthesis. In *Proc. Design, Automation and Test in Europe Conf. (DATE2001)*. 620 – 625.

Zhang, Y., Hu, X., and Chen, D. Z. 2002. Task Scheduling and Voltage Selection for Energy Minimization. In *Proc. IEEE 39th Design Automation Conf. (DAC02)*. 183–188.