Provably Minimal Energy using Coordinated DVS and Power Gating

Nathaniel A. Conos, Saro Meguerdichian, Foad Dabiri, and Miodrag Potkonjak
Computer Science Department
University of California, Los Angeles
{conos, saro, dabiri, miodrag}@cs.ucla.edu

***Abstract*—Both energy and execution speed can be greatly impacted by clock and power gating, nonlinear voltage scaling, and leakage energy. We address the problem of coordinated power gating and dynamic voltage scaling (DVS) to minimize the overall energy consumption of an application under user-specified timing constraints. We prove that a solution provided by our convex programming formulation that uses at most two versions of hardware, where each version uses its own constant voltages, is optimal. Comprehensive evaluation of the new approach demonstrates energy improvements over traditional DVS and DVS and power gating techniques by factors of 1.44X–2.97X and 1.44X–2.82X, respectively.**

I. Introduction

There is a wide consensus that energy is a premier design and operational metric for various computing systems ranging from data centers and desktop computers to smart phones and sensor networks. However, energy minimization in these modern and pending systems for time sensitive tasks is an increasingly complex and intractable problem because of several interwoven degrees of freedom, such as supply voltage management, leakage currents, and unit clock/power gating. For example, dynamic voltage scaling (DVS) is by itself complex because the various components (e.g. functional units and cache memory) have highly differing energy-speed of execution trade-offs. Therefore, the overall system execution speed is a nonlinear function with discontinuities.

Additional challenges for energy minimization include the significant overhead (ranging from hundreds to even thousands of clock cycles) induced by voltage and frequency adjustments. Leakage energy alone depends on several factors, including the allocated hardware, supply voltage, processs variation, and temperature [6][7]. It is well known that clock and power gating are very effective energy saving techniques with relatively low time and energy overheads. However, careful power gating and DVS settings are still necessary when considering the entire task schedule, since their aggregate overheads may greatly impact the execution time.

We address the problem of minimizing the total energy required to complete a computational task within a specified allocated time. We have developed a new approach that combines the effectiveness of static power gating and DVS techniques using a convex programming-based procedure. It has been a long standing common wisdom that a single voltage should be used for the execution of a task. It was proven that this strategy is optimal when the energy-speed dependency is convex. Recently, it was demonstrated that when the relationship is non-convex, it is often more advantageous to employ multiple voltages [4]. However, unless there are drastic differences in speed and speed-energy products for a given application, the benefits of the technique are limited. We create these differences by realizing N versions of the pertinent hardware using static power gating.

The essence of the approach is illustrated in Figure 1, which shows energy-speed trade-offs for eight versions of hardware used to realize a synthetic application. In Figure 1a, there are eight points that correspond to eight power gating options of the available hardware, operating at a single voltage. The base hardware platform corresponds to the maximal-resource hardware allocation 8 (blue hexagram). Allocations 7 (purple pentagram) through 1 (blue square) are allocations with decreasing hardware resources. If our time constraint, for example, requires a clock rate of 650 MHz, the best individual configuration (without utilizing DVS or power gating techniques) is allocation 8 at nominal supply voltage, requiring 4 mJ, as shown in Figure 1a.

In Figure 1b, there are eight sets of three points that correspond to three operational voltages of each hardware allocation. In this case, for the same timing constraint, the optimal single configuration, where a configuration is comprised of a hardware allocation and operating voltage utilizing DVS, is allocation 6 (yellow right-arrow) set to the highest voltage, consuming 3 mJ (25% savings). Note that the selected configuration is the lowest energy point that exceeds the speed requirement. Figure 1c demonstrates how power gating can be used to reduce the energy consumption by turning off the unused hardware after the completion of the application. This can be done because the speed of the chosen configuration is faster than required. In this example, the energy consumption is now 2.7 mJ (32.5% savings).

Finally, Figure 1d demonstrates our new approach that uses coordinated DVS and power gating to provide provably minimal energy. The first step is to find the convex enclosure of the energy-speed points, consisting of piecewise linear segments such that all points are either on the enclosure or above and to the left of it. The intuition behind computing the convex enclosure is that we can use a combination of the two closest points on the enclosure that surround the target speed requirement to execute the task using the minimum amount of energy. In this example, the highest-voltage setting of allocation 6 (yellow right-arrow) can be used for 42% of

978-3-9815370-2-4/DATE14/©2014 EDAA IEEE

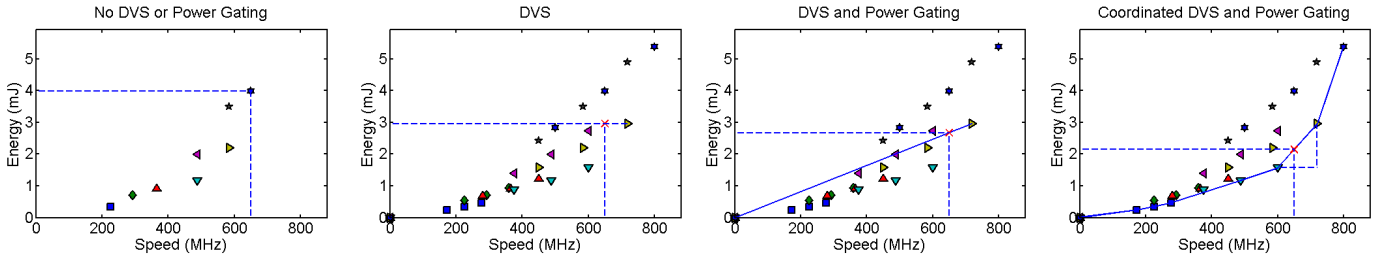


Fig. 1: Motivating example with 8 hardware allocations at 3 voltages: (a) no DVS or power gating; (b) DVS but no power gating; (c) DVS and power gating; and (d) our approach, coordinated DVS and power gating.

the required time ($\frac{650-600}{720-600}$), followed by a context switch to allocation 4 (teal down-arrow) for the remaining 58% of the execution time. This approach requires 2.2 mJ (45% savings). Note that in this case changing the voltage between the two optimal points was not required, but in general DVS can be utilized, if needed, to switch to a different voltage.

It is crucial to note that switching between allocations is realized by power gating subsets of microarchitectural units to, for example, reduce the number of units or the cache sizes. Thus, using n sets of power gating circuitry, 2^n hardware allocations can be realized. Our coordinated DVS and power gating procedure is surprisingly effective, since schedules utilizing only two voltages and two versions of an implementation (hardware allocation) are used and can be determined at compile time. There is only one context switch, where either a subset of microarchitectural units are power gated to realize a different allocation, or voltage is scaled, or both. Therefore, the additional storage overhead is small and the overhead for task management is negligible.

To summarize, the overall flow of the optimization proceeds as follows: (1) create the gating structure; (2) characterize the application of interest; (3) calculate the DVS impact; (4) calculate the convex enclosure; and (5) select the two best allocations at the two best voltages (two configurations total).

II. Related Work

Dynamic voltage scaling methods have been proposed since more than two decades ago. They address energy minimization by only altering the supply voltages. Those methods cover various sets of scenarios and system specifications ranging from continuous to discrete supply voltages to achieve energy efficiency, but these have mainly focused on dynamic power consumption [11][17]. Researchers have also studied the DVS techniques in conjunction with peripheral management and further generalized the method to multiprocessors [9][18].

Leakage energy consumption and its impact on the overall system energy dissipation have significantly grown because of the continuous scaling of CMOS to miniature feature sizes. In fact, for some 35-nm processes, leakage could potentially be even larger than dynamic energy [5][6][13]. Therefore, a MIT group has studied techniques to address leakage energy, where leakage current is modeled at different levels of abstraction [12]. Furthermore, DVS has been extended to control threshold voltages (V_t) and therefore leakage current through adaptive body biasing [16]. Various methods for simultaneous supply voltage and threshold voltage scaling are also covered [1][10].

Power gating at the microarchitectural level has been proposed to achieve further reductions in leakage, by gating units when idle periods of 10 cycles or more are detected [8]. They achieve significant leakage savings, but only consider one or two units. Furthermore, the impact of leakage current on energy and the usage of various hardware have led to more general and non-convex energy-speed models [12][14].

Our approach differs from all previous techniques because we assume non-linear tradeoffs between energy and speed of execution with an arbitrary number of discontinuities. Furthermore, we show that partial power gating and DVS can be combined in a provably optimal way to minimize energy.

III. Preliminaries

A. Energy & Delay Model

We adopt the energy and delay models used by Dabiri et al. [4]. For a hardware with M_s resources, switching capacitance is a function of the resources used, $C(M_s)$, where one iteration of the schedule takes R_s clock cycles, and the clock period is $T = \frac{1}{f}$. Since energy is the integral of the instantaneous power over time and each iteration takes R_sT seconds, the total energy ($E_{total(s)}$) of this schedule for one iteration of the schedule is:

$$\begin{aligned} E_{total(s)} &= R_sTP_{total(s)} \\ &= R_sT[\tfrac{1}{2}\alpha CV_{dd}^2 f + V_{dd}I_{sub}] \end{aligned} \quad (1)$$

In the above equation, $P_{total(s)}$ is the total power consumption, the sum of dynamic (charging the capacitive load $C(M_s)$) and static (leakage) power, where I_{sub} is the subthreshold current commonly used for approximating leakage. In the optimizations done in this paper, for each schedule we utilize the average energy per clock cycle $\frac{E_{total(s)}}{R_s}$ versus the schedule speed R_s/T for one iteration.

The delay of CMOS based processing elements can be stated as:

$$d = K \times \frac{V_{dd}}{(V_{dd} - V_t)^\alpha} \quad (2)$$

where K and α are technology-dependent parameters. Equation 2 in its general form is used to model delay and its changes with respect to operation frequency and supply voltage.

B. Multi-Allocation Architecture

Our objective is to minimize the energy consumption per task or collection of tasks, while satisfying the system- and

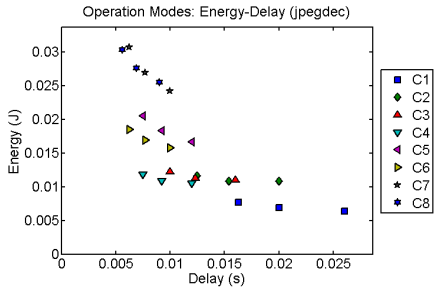


Fig. 2: Energy-delay points for different configurations (8 allocations and 3 voltages) for the jpegdec benchmark.

application-imposed constraints, such as latency or throughput. Equation 1 suggests that there are potentially two main variables that can be controlled to alter system configurations. One is the hardware capacitance (C), and the other component is the supply voltage (V_{dd}), which leads to DVS techniques.

For the first component, we utilize processing elements called *hardware allocations*, where each allocation effectively has a different $C(M_s)$ (refer to Equation 1). Allocations can be individual processing units or can be constructed using a subset of computational components from a larger computing platform. Note that with n power gating hardware positions, 2^n different hardware allocations can be created.

We are given (or designed) a set of hardware allocations, where each utilize different sets of hardware components. An allocation v_i is identified by a unique tuple, $v_i \equiv (e_i, d_i, s_i, R_i)$, where e_i is the energy per clock cycle of an allocation for a given schedule; d_i is the allocation's delay (latency) for a scheduling cycle; s_i is the allocation's *effective* speed; and R_i is the number of clock cycles to process a given schedule on the allocation.

The second effective variable is the supply voltage (V_{dd}). Energy optimization via dynamic or static voltage scheduling and scaling is a very well studied problem. What distinguishes this paper from previous work in DVS is that our methodology is a hybrid of hardware resource management and voltage scaling. The former corresponds to the fact that we potentially utilize different hardware allocations in the scheduling (via gating), while the latter indicates that each of these allocations can have multiple discrete supply voltage choices.

C. Configurations

We assume to have M hardware allocations where each can operate under K discrete supply voltages $V = \{V_{dd_1}, ..., V_{dd_K}\}$, which then leads to $N = M \times K$ *configurations*. Each configuration is represented as a hardware allocation-supply voltage pair, $m_i = (v_j, V_{dd_k})$. To each configuration we shall assign an effective speed (computed by Equations 3 and 2) as well as a value for energy per clock cycle (derived from Equation 4), where we represent a configuration as $m_i(v_j, V_{dd_k}, s_i, e_i)$. As can be observed, the configurations have a similar representation as the hardware allocations, which is natural since each hardware allocation is in fact a configuration under a given supply voltage.

Assume a configuration m_x is the fastest configuration in the sense that for a given schedule it requires R_x clock cycles, where $R_x = min(R_1, R_2, ..., R_N)$ and N is the number of configurations. This configuration is called the *base configuration* and its speed s_x is said to be the clock frequency f_{clk}. We normalize other configurations' specifications to the base configuration and define the *effective speed*, s_i, for configuration m_i as follows.

The latency d_i of m_i for a schedule which takes R_i clock cycles is $d_i = R_i/f_{clk}$. Consequently, if we assume that the schedule requires R_x clock cycles but the operational frequency (speed) of the configuration is s_i (not f_{clk}), the equivalent or effective speed of configuration m_i is:

$$s_i = f_{clk} \times R_x/R_i, \tag{3}$$

which results in the same delay, d_i. Under this normalization, switching across configurations is virtually equivalent to changing the operation frequency of the configuration under the assumption that the required number of clock cycles (R_i) remains the same. The energy per clock cycle of a configuration is also normalized with R_x, yielding:

$$e_i = \frac{R_i}{R_x} V_{dd} I_{sub} T + \frac{R_i}{R_x} \frac{1}{2} \alpha C V_{dd}^2. \tag{4}$$

Figure 2 shows energy consumption vs. latency for the jpeg-decoder benchmark. In this graph, we have used 8 hardware allocations, where for each allocation we show three supply voltages. We apply the normalization in Equation 3, which leads to energy-speed points shown in Figure 3. Details of the configurations (hardware allocation-supply voltage pairs) are described in Section VI.

IV. Problem Formulation

A. Optimization Objective

Our key objective is to use multiple configurations to process a given task such that the total energy consumption of the system is minimized, while the processing is completed prior to a specified deadline. A scheduling output can be represented as an ordered series of configurations and the length of time each configuration is scheduled for operations: $\Psi =< (m_1, t_1), (m_2, t_2), (m_k, t_k) >$, where $M_r = \{m_1, v_2, ..., m_N\}$ are the N configurations as defined in Section III-C, r is the scheduled task, and the duration for which each configuration m_i is active is t_i. Therefore, the total energy consumption for this schedule and optimization objective can be defined as:

$$E_\Psi = \int_0^T P(\xi(t))dt = \sum_0^R e(s_i)\Delta R \tag{5}$$

$$minimize(E_r), s.t. D_r = \sum_{i; v_i \in V_r} t_i \leq T \tag{6}$$

where D_r is the processing delay/latency for the schedule Ψ and T is the deadline.

B. Configuration Switching Overhead

Switching configurations has potentially two sources of overhead: (1) power gating overhead caused by switching across hardware allocations; and (2) voltage scaling overhead. The overhead presents itself in both energy and delay.

1) Power-Gating Overhead: Power gating is done by placing a suitably sized header or footer transistor for a circuit block. The amount of switching energy in the header device (E_{header}) and the number of cycles needed to power gate a macro before reaching the break-even point ($N_{breakeven}$) are computed as [8]:

$$E_{header} = 2C_{header}V_{dd}^2 \approx 2W_H\frac{1}{2}C_{switching}V_{dd}^2. \quad (7)$$

$$N_{breakeven} = 2\frac{1}{2L\alpha}\sqrt{\frac{mV_tW_H}{V_{dd}DIBL}(1+2\frac{C_{supply}}{C_{switching}})}, \quad (8)$$

Parameter definitions are presented by Hu et al.; $N_{breakeven}$ is shown to be about 10 clock cycles [8].

Assuming that the ratio of the header device W_H is constant for all the modules, the timing overhead of power gating remains the same for the different schedules. However, the energy overhead is a monotonic function of the number of components in the schedule. Using Equations 7 and 8, this overhead can be modeled as:

$$\epsilon_{ij} = f(v_i, v_j) \propto |N_i - N_j|, \quad (9)$$

$$\delta_{ij} = g(v_i, v_j) = \delta. \quad (10)$$

Equation 9 indicates that the energy overhead as a result of power gating is a function of the difference in the resources in hardware allocations v_i and v_j, whereas the delay overhead could be assumed to be a constant value.

2) Voltage Scaling Overhead: Switching the supply voltage from V_{dd_i} to V_{dd_j} creates overheads in energy (ϵ_{ij}) and delay (δ_{ij}), which is represented by switching from configuration i to j for each respective case. We use results from Andrei et al. [1] and Martin et al. [16] to model these overheads as:

$$\epsilon_{ij} = C_r|V_{dd_i} - V_{dd_j}|^2 + C_s|V_{bs_i} - V_{bs_i}|^2 \quad (11)$$

$$\delta_{ij} = max(p_{V_{dd}}|V_{dd_i} - V_{dd_j}|, p_{V_{bs}}|V_{bs_i} - V_{bs_i}|) \quad (12)$$

where C_r and C_s are constants for power rail capacitance and substrate-well capacitance, respectively. Note that when a configuration switch occurs, potentially both supply voltage V_{dd} and body-bias voltage V_{bs} change. The delay overhead caused by supply voltage and bias voltage changes are proportional to $p_{V_{dd}}$ and $p_{V_{bs}}$, respectively, and the larger delay of these two will be the overall delay overhead.

V. Optimal-Energy Scheduling

In this section, we derive a set of properties of our formulation and an optimal N-configuration schedule that leads us to the proposed methodology. This sections extends the work in [4] by incorporating both hardware allocation as well as operating supply voltage.

First, we illustrate methods for emulating any virtual operating speed using a mixture of configurations. Consider a simple example where there are two configurations m_1 and m_2. For the sake of simplicity, we have omitted the configuration switching overheads in this example. Recall that because the optimal solution uses at most two configurations, requiring a maximum of one configuration switch, this overhead is negligible. In order to run the system at speed s^* ($s_1 \leq s^* \leq s_2$) for a given interval $[a, b]$, we first use

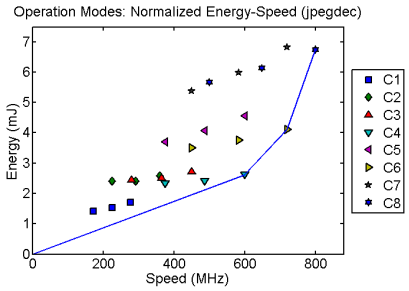


Fig. 3: Energy-speed points for different configurations for the jpegdec benchmark. Note that the speed is *normalized* as described in Equation 3.

configuration m_1 in speed s_1 for t_1 seconds and m_2 with s_2 for t_2 seconds, where t_1 and t_2 are:

$$t_1 = \frac{s_2 - s^*}{s_2 - s_1} \times (b-a), t_2 = \frac{s^* - s_1}{s_2 - s_1} \times (b-a) \quad (13)$$

where s^* is the weighted average speed when the system is run at speeds s_1 and s_2, indicating that in the duration of $[a, b]$ the system was operating at the virtual speed of s^*. In the presence of switching overheads, the target speed would be $s^*_\delta = \frac{b-a}{b-a-\delta}s^*$. In this scenario, if $s^*_\delta < s_2$, then:

$$\mathrm{t}_1 = \frac{s_2 - s^*_\delta}{s_2 - s_1} \times (b-a), t_2 = \frac{s^*_\delta - s_1}{s_2 - s_1} \times (b-a) \quad (14)$$

Otherwise, $s^*_\delta \geq s_2$ indicates that in order to compensate for switching delay overhead, effective speed should be more than s_2. Therefore, there is no need to use two configurations and the processing can happen only with configuration m_2 for the duration of $b - a$:

$$\mathrm{t}_1 = 0, t_2 = \frac{s^*}{s_2} \times (b-a) \Rightarrow s^* = s_2, t_2 = b \quad (15)$$

A. Convex Energy-Speed Curve

Given two configurations m_i and m_j with effective speeds of s_i and s_j, $\mathcal{E}_{ij}$ is defined to be the the minimum energy per clock cycle for a given virtual speed s^*. Thus, when considering configuration switching overhead, $\mathcal{E'}_{ij}(s^*)$ is simply the minimum energy consumed between its representative virtual speed or by a single configuration, as described by:

$$\mathcal{E}_{ij}(s^*) = \frac{e_j - e_i}{s_j - s_i}(s^* - s_i) + e_i, \quad (16)$$

$$\mathcal{E'}_{ij}(s^*) = min(\mathcal{E}_{ij}(s^*) + \Delta e_{ij}, e_i). \quad (17)$$

Using this notion of a continuous energy-speed definition, we form a bounding curve on the energy-speed points of the configurations. This curve is convex when switching overhead is ignored and has a convexity property with switching overhead under some conditions which are presented below. Figure 3 shows this continuous energy-speed curve, which will be the target operation space. When considering switching overhead, the optimal bounding curve would have vertical shifts (proportional to its energy overhead) and horizontal segments. Horizontal segments are cases where running at a higher speed than s* (e.g. a single configuration) would be more energy efficient than switching between two surrounding configurations due to the overhead. Note that the optimality of the proposed algorithm still holds by just adding the energy (ϵ_{ik}) and speed (δ_{ij}) overheads to the proof of Theorem 5.1.

TABLE I: Allocation parameters.

Allocation	IDC	LSU	ALU	MUL	L1	L2
1	2	2	2	1	16KB	*none*
2	2	4	2	1	16KB	32KB
3	4	4	4	2	16KB	32KB
4	4	8	4	2	16KB	32KB
5	8	8	8	2	32KB	64KB
6	8	16	8	4	32KB	64KB
7	16	16	8	4	32KB	64KB
8	16	32	16	8	64KB	128KB

TABLE II: Power (W) at max V_{DD}.

Allocation	IFU	LSU	MMU	ALU	MUL	L2
1	0.33	0.17	0.05	0.19	0.89	*none*
2	0.33	0.17	0.09	0.19	0.89	0.37
3	0.38	0.22	0.10	0.25	1.19	0.37
4	0.38	0.22	0.10	0.25	1.19	0.37
5	0.63	0.37	0.37	0.82	1.19	0.64
6	0.63	0.37	0.37	0.82	3.85	0.64
7	0.88	0.55	0.91	0.82	3.85	0.64
8	1.02	0.64	0.91	1.48	6.92	1.10

B. N-Configuration Scheduling Algorithm

Theorem 5.1: There is an optimal N-configuration scheduling where only two or less configurations are used throughout the execution of a task

Proof: Assume there is an optimal configuration scheduling Ψ which uses k different configurations where $k > 2$. Furthermore assume m_1, m_2, and m_3 are three consecutive configurations in terms of energy consumption in the schedule and each runs for a duration of t_1, t_2, and t_3 seconds respectively. We will show that there is another schedule Ψ^* that can be derived from Ψ which uses one less configuration and $E_{\Psi^*} \leq E_{\Psi}$ while $D_{\Psi^*} \leq D_{\Psi}$. We omit the details for brevity but there are two cases to consider in the proof:

- $\mathcal{E}_{13}(s_2) \leq e_2$: This shows that by removing m_2 and only using m_1 and m_3, total energy consumption will be reduced since s_2 can be virtually achieved using equation 13, which leads to energy consumption compared to $\mathcal{E}_2$;
- $\mathcal{E}_{13}(s_2) > e_2$: This scenario itself is divided into three cases: $s^* < s_2$, $s^* > s_2$, or $s^* = s_2$, where s^* is the effective speed of the schedule when switching between the three configurations. If $s^* < s_2$, it is trivial that s^* can be created using only m_1 and m_2 with less energy consumption. Similar arguments hold for $s^* > s_2$ and if $s^* = s_2$ and it is evident that only m_2 should have been used for the whole execution of the task.

Therefore, we can reduce the number of configurations in the schedule by one and still use less or equal amount of energy. The same method is applied recursively till at most two configurations remain in the schedule. ■

An immediate observation follows that for a given feasible schedule, the average speed of the system, s^* has a lower bound of R_x/T where T is the schedule deadline and R is the required number of clock cycles when the fastest configuration is used. We call this speed the *critical speed*. The idea behind the methodology is to utilize the maximum slack available and run the system at the lowest speed possible, s^*, to minimize the energy consumption. Algorithm 1 summarizes the optimal scheduling for one task. For multiple tasks, we use the results from [19] and apply the same scheduling with the observation that for a critical speed, s^*, the schedule and configurations are found using Algorithm 1. Note that the scheduling is the implicit result of Algorithm 1.

Algorithm 1 N-Configuration Scheduling: Single Task.

1: Find the critical speed: $s^* = R_x/T$;
2: Find i and j such that $\mathcal{E'}_{ij}(s^*)$ is minimized (binary search);
3: Use Equation 15 to find the configuration and schedule times for m_i and m_j;

Theorem 5.2: Algorithm 1 results in the minimum energy consumption per frame while meeting all hard deadlines.

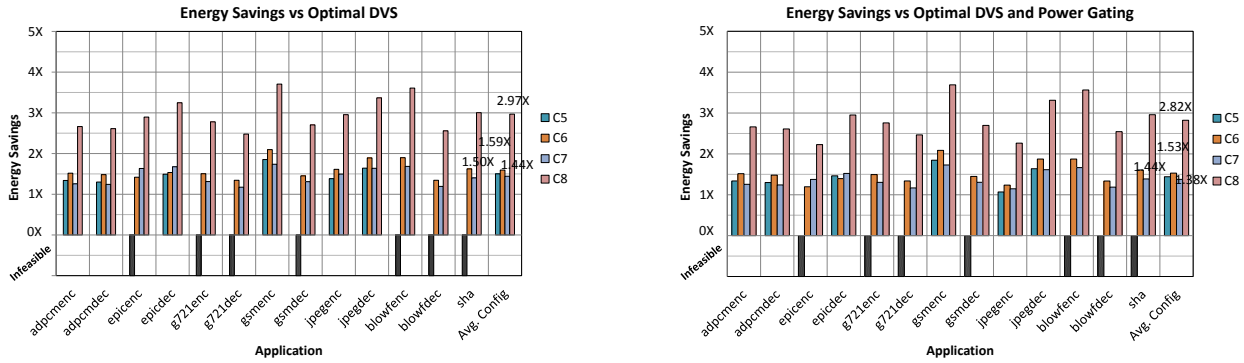
Proof: From 5.1 we conclude that at most two configurations are needed to find the minimum energy consumption. Also, from step 2 of the Algorithm 1, the minimality of energy consumption for 2-configuration schedules is guaranteed. ■

VI. Experimental Results

We used the SimpleScalar-ARM simulator [3] to generate single-threaded ARM7-ISA cycle accurate traces. Resources considered are: 1) instruction fetch units (IFU); 2) load-store units (LSU); 3) arithmetic logic units (ALU); 4) multipliers (MUL); and 5) level 1 and 2 instruction/data caches (L1/L2) (Table I). Our power model uses 45-nm parameters included in McPAT [15]. We extract the power values for each modeled resource for each configuration (Table I). Configuration energy (ϵ_{ij}) and delay (δ_{ij}) overheads are computed using representative functional unit load, rail, and substrate capacitances using equations in Section IV. We cover eight hardware allocations and enable five discrete supply voltages (0.7, 0.8, 0.9, 1.0, and 1.1V). Due to space constraints, Table II only lists the hardware allocations' combined dynamic and static powers at the maximum supply voltage. We update the Wattch [2] model with these values to generate total energy and runtime values for each benchmark at each configuration.

We performed our evaluations on 13 different benchmarks, considering 40 configurations comprised of 8 hardware allocations and 5 different supply voltages per allocation. We compare our optimal coordinated DVS and power gating approach against a) DVS alone and b) DVS and power gating. Recall that both of these approaches use only a single allocation and voltage, while our approach uses up to two allocations and two voltages (two configurations) that achieve the optimal energy for a given delay constraint.

Figures 4a and 4b illustrate the energy savings achieved in comparison to these scenarios. The x-axis represents the application and the y-axis is the normalized energy savings. For each application, there are 8 columns corresponding to the different allocations. For applications epicenc, g721enc, g721dec, gsmdec, blowfenc, blowfdec, and sha, allocation C5 could not be used to execute the application by the given deadline. In this case, a black bar is shown below the axis to indicate that the base case (either DVS or DVS and power gating) does not have a feasible solution and therefore no energy comparison can be made. Allocations C1-C4 are omitted from the results because they could not be used by the DVS and DVS and power gating approaches to execute any applications within the allotted time. This is to be expected, since these allocations have very limited hardware resources.



(a) (b)

Fig. 4: Energy savings of our coordinated DVS and power gating approach vs. (a) DVS and (b) DVS and power gating. Note that for a subset of applications, hardware allocation C5 is not fast enough to meet the required deadline. Furthermore, allocations C1-C4 are omitted from the results because they could not meet the deadline for any application.

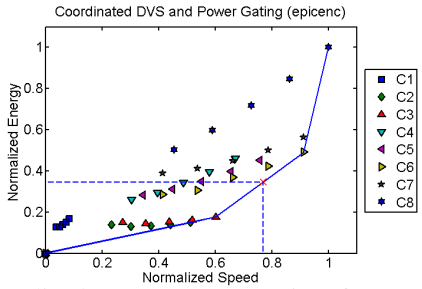


Fig. 5: Normalized energy consumption for a given delay constraint for the epicenc benchmark using our coordinated DVS and power gating approach.

Furthermore, Figure 5 shows the given deadline and result for our approach for the epicenc application. It is clear from the figure that although hardware allocations C1-C5 cannot meet the deadline independently, the optimal result uses a combination of allocations C6 and C3, whose highest-voltage configurations are the surrounding points of the target speed on the convex enclosure. For each allocation, the DVS-only approach would consume energy equivalent to the energy consumption of the lowest-energy configuration to the right of the target speed. In the DVS and power gating approach, the same configuration would be selected but powered off after completing the task.

VII. Conclusion

We have developed a new approach for energy minimization under timing constraints that combines the effectiveness of power gating and dynamic voltage scaling (DVS) in a coordinated manner. We use a convex programming procedure to optimally solve the problem without placing any restrictions on the energy-speed of execution relationship or the voltage step values. The technique is highly practical; on standard benchmarks, our method results in an average savings of 1.44X–2.97X and 1.44X–2.82X with respect to the best DVS and DVS and power gating solutions, respectively.

References

[1] A. Andrei et al., "Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems," *DATE*, pp. 518–523, 2004.

[2] D. Brooks et al., "Wattch: a framework for architectural-level power analysis and optimizations," *SIGARCH*, vol. 28, pp. 83–94, 2000.

[3] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," *SIGARCH*, vol. 25, pp. 13–25, 1997.

[4] F. Dabiri et al., "Energy minimization for real-time systems with non-convex and discrete operation modes," *DATE*, pp. 1416–1421, 2009.

[5] Conos, N.A. et al., "Gate sizing in the presence of gate switching activity and input vector control," *VLSI-SoC*, pp. 138–143, 2013.

[6] Conos, N.A. et al., "Maximizing yield in Near-Threshold Computing under the presence of process variation," *PATMOS*, pp. 1–8, 2013.

[7] Conos, N.A. and Potkonjak, M., "A temperature-aware synthesis approach for simultaneous delay and leakage optimization," *ICCD*, pp.316–321, 2013.

[8] Z. Hu et al., "Microarchitectural techniques for power gating of execution units," *ISLPED*, pp. 32–37, 2004.

[9] C.-M. Hung et al., "Energy-efficient real-time task scheduling for a DVS system with a non-DVS processing element," *RTSS*, pp. 303–312, 2006.

[10] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," *ISLPED*, pp. 192–202, 1998.

[11] R. Jejurikar and R. Gupta, "Energy aware task scheduling with task synchronization for embedded real time systems," *CASES*, pp. 164–169, 2002.

[12] J. Kao et al., "Subthreshold leakage modeling and reduction techniques," *ICCAD*, pp. 141–148, 2002.

[13] H.-S. Kim, "Impact of scaling on the effectiveness of dynamic power reduction schemes," *ICCD*, pp. 382–387, 2002.

[14] R. Kumar et al., "Heterogeneous chip multiprocessors," *Computer*, vol. 38, no. 11, pp. 32–38, 2005.

[15] S. Li et al., "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," *MICRO*, pp. 469–480, 2009.

[16] S. M. Martin et al., "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads," *ICCAD*, pp. 721–725, 2002.

[17] P. Rong and M. Pedram, "Power-aware scheduling and dynamic voltage setting for tasks running on a hard real-time system," *ASP-DAC*, pp. 473–378, 2006.

[18] T. Wei et al., "Online task-scheduling for fault-tolerant low-energy real-time systems," *ICCAD*, pp. 522–527, 2006.

[19] F. Yao et al., "A scheduling model for reduced CPU energy," *FOCS*, pp. 347–382, 1995.