Coordinated and Adaptive Power Gating and Dynamic Voltage Scaling for Energy Minimization

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

***Abstract*—Customization and adaptation have emerged as the most effective paradigms for energy minimization. We employ these paradigms to address coordinated power gating and dynamic voltage scaling for energy minimization of real-time tasks in both application-specific and programmable systems. For a given hardware platform, the first task in adaptation is to determine which hardware allocations and supply voltages should be used to execute a given set of tasks. In the case where tasks induce constant capacitance and are of sufficient length to render power gating and voltage scaling overheads negligible, we obtain the provably optimal solution using a combination of convex enclosure and convex optimization. Our next goal is to relax these two assumptions to consider overheads and non-uniformity in capacitance by subdividing each task into multiple subtasks at a fine granularity. In this case, we use another dynamic programming formulation to find a solution which is optimal in most practical cases. We have also developed a dynamic programming-based approach to minimize overheads of configuration switching.**

I. Introduction

There is a wide consensus that energy efficiency is a design metric of paramount importance. It is, for instance, important for computers in data centers in order to minimize cooling cost that is measured to be in the tens of millions of dollars. It is crucial for mobile wireless devices such as phones and tablets in particular with a pending transition from voice to data traffic (e.g. IPTV and complex video games). Energy minimization is rarely an easy task but it is even more demanding in systems that are subject to real-time constraints.

Therefore, it is not surprising that a tremendous amount of industrial and academic effort has been dedicated to energy minimization, most often under the umbrella of low power research. The initial energy models were simple from the optimization point of view and included only switching energy as impacted by supply voltage, frequency, and switching activity. The dependency between the speed of execution and the energy was convex and hence simple for optimization. Finally, initially only strict real-time constraints were considered.

Today, in addition to switching energy, several types of leakage energy are increasingly important. Systems are complex in the sense that the energy of different components scales differently with the supply voltage. Also, the most effective techniques such as power or clock gating result in highly non-linear speed of execution to energy trade-offs. Dynamic voltage scaling techniques for energy minimization now provide rather limited improvements. From one side currently used supply voltages are already greatly reduced as dictated by finer feature sizes. From another side process variation results in some gates having rather high voltage thresholds. Therefore, the gap between the maximal and minimal feasible voltage is small and continues to shrink. As a ramification, the need for emphasis on different energy minimization mechanisms is well established.

The new most popular applications such as mobile phone calls, WWW surfing, and movie watching on smart phones are subject to new types of real-time constraints [13]. They also have segments with sharply different amounts of parallelism and requirements for different types of resources. Therefore, the important problem of determining ideal hardware configurations and run time allocations must be addressed in a synergistic fashion across several levels of system abstraction and design phases. For example, identifying and choosing hardware allocations that maximize energy efficiency on given a set of functional units with very different energy-speed trade-offs across a set of expected applications early in design is of crucial importance, since the choices made have significant impact on available run time configurations later. Furthermore, once a set of available system configurations is defined, it is important in the later design stages to identify the suitable time allocations for applications, since they can be fine-tuned to their respective run time configurations. The decisions made in these steps can have dramatic effect on power management schemes, which are often managed at the operating system level, such as through the widely accepted Advanced Configuration and Power Interface (ACPI) utilizing dynamic voltage scaling and clock and/or power gating schemes [1].

In summary, for modern wireless mobile applications the standard energy minimization techniques have limited effectiveness, and conceptually new energy minimization techniques are required at both the architectural and operating system levels. These techniques are associated with much more complex optimization strategies that are highly non-linear, non-convex, and with a large number of discontinuities.

A. Problem Formulation

We focus on one real-time synthesis and one energy management problem that can be informally stated in the following highly simplified way. We start with the low energy synthesis problem. A task and a computational platform are given. The computational platform can be configured in a large number of ways and profiling of the task is conducted for each platform. The goal is to select a user-specified number of configurations such that the task (application) consumes

minimal energy while being completed within a user-specified total execution time. No restriction on the speed-energy curve are imposed. Our basic operating system-level problem is to find the allocation of an available time for execution of a given task on each of the allocated hardware configurations, so that the total used energy is minimized.

These two generic problem formulations are the starting point for several more complex and practically important versions. For example, we can address formulations where multiple applications that may be subject to uncertainty are considered. Or we may consider the situation where each task is composed of a number of sub-tasks with unique speed-energy curves on different hardware configurations.

II. Related Work

In the early '90s the feature size of integrated circuits was 1 micron and the supply (V_{dd}) and threshold (V_{th}) voltages were 5 V and around 1 V, respectively. The high gap between V_{dd} and V_{th} and the quadratic dependence between the energy and the supply voltage enabled improvements of more than an order of magnitude and suggested a paradigm where the goal is to transform the pertinent computation in such a way that first the maximal speed-up is created, and consequently it is used for the reduction of the supply voltage and therefore energy minimization [2]. The essential background information is that the switching power (P) is equal to $C_{eff} \cdot V_{dd}^2 \cdot s$, where s is the speed of execution and $s = k \cdot (V_{dd} - V_{th})^2/V_{dd}$, C_{eff} is effective switching capacitance, and k is a technology and integrated circuit dependent constant. Note that only dynamic (switching) energy was considered because at that time the static (leakage) component was negligible.

Soon it was realized that the same paradigm could be used as a basis for the minimization of energy in a more complex computational model. Yao, Demers, and Shenker considered a set of aperiodic jobs where any point on a continuous convex trade-off energy-speed curve is available and where there is no overhead for changing the speed [3]. Ishihara and Yasuura considered a more realistic energy-speed model where only a discrete subset of trade-off points is available and under the assumption of a negligible speed change overhead [4]. They proved that in this case the use of two consequent options that bound the required optimal speed produces the optimal solution. Vigorous research efforts along this line eventually resulted in a fully polynomial approximation algorithm with arbitrarily tight approximation [5].

At the turn of the century it was realized that the energy-speed relationship is no longer convex due to several reasons including different operational speeds of different components (e.g. memory vs. datapath) and rapidly increasing leakage current in deep submicron technologies [7]. In the current 22 nm technology it forms almost one half of the overall energy consumption. The initial observation was that by powering down one can save a very significant amount of energy. Soon, in addition to active mode, idle, standby, and sleep were considered as shutdown states each with lower energy and longer activation time. Irani et al. have developed a 3-approximation off-line algorithm to minimize dynamic and static energy using as a starting point the algorithm by Yu, Demers, and Shenker [6]. Lee, Reddy, and Krishna introduced the procrastination concept, where the system enters or stays in a shutdown mode even when there are pending tasks in order to reduce leakage energy [8]. The procrastination algorithms have attracted significant attention [9][10]. From another perspective, Chandrakasan et al. showed that by trading silicon area, power consumption can be reduced while maintaining throughput, for example by replicating hardware and reducing the operating voltage [11].

Another important class of energy minimization problems is a set of dual problems to already stated ones, where the goal is to minimize the allocation cost of the hardware platform under energy and/or timing constraints [15][16]. They proposed heuristics and approximation algorithms for this computationally difficult variant of our problems. Recently, Dabiri et al. presented surprising results that in a sense ultimately addressed optimization when the energy-speed trade-off has an arbitrary dependency [17]. Regardless of the shape and form of the dependency, they showed one can always provably optimally minimize the total energy of any task using at most two actual speeds by using computational geometry concepts. An example platform is shown in [18].

Multiple supply and/or threshold voltages are another popular set of energy minimization techniques [19][20][30]. However, both multiple V_{dd}/V_{th} and dynamic voltage scaling techniques have decreasing efficiency as technology progresses. Two main reasons are that feature scaling drastically reduced not just supply and threshold voltages but also their gap. Also, process variation causes threshold voltage to become subject to a relatively wide distribution, and the threshold voltage must be higher than the highest threshold voltage of any transistor [7][12][14]. Therefore, although dynamic voltage scaling is still important, power gating may now enable much better improvements.

In the last decade, power gating was recognized as an effective way to reduce leakage energy [21][22][23][24][25][35]. Our goal is to provide an impetus for further development of this line of optimization by showing how a simple but rich power gating structure can be adaptively used to facilitate energy minimization in modern and pending systems. For the sake of brevity and due to space limitations we do not cover energy minimization in distributed systems such as wireless phones, sensor networks, and data centers [2][27] and simultaneous thermal management and energy minimization. Furthermore, this work primarily focus on offline hardware allocation schedules, and note that techniques can be readily applied in the online domain using statistical means, such as with [6][26][33].

III. Preliminaries

We begin by describing a critical set of technical preliminaries, including creation of hardware allocations via power gating; detailed profiling of applications; a previously proposed approach for executing a single application at a specified speed with minimal energy; and our simulation platform and energy, delay and configuration switching overhead models.

A. Multi-Allocation Architecture

In this section, we describe a method of employing power gating techniques to create processing elements called *hardware allocations*. A hardware allocation can be defined as a

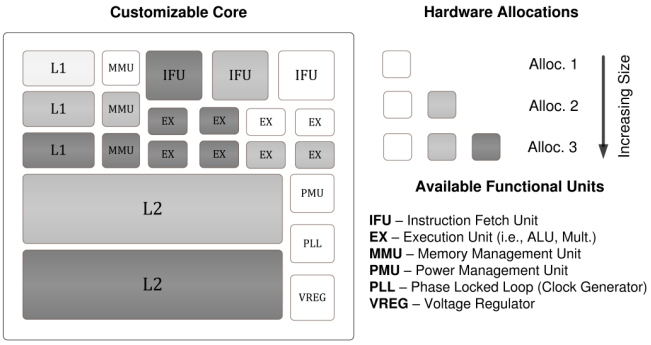


Fig. 1: Hardware allocation example showing three hardware allocations: 1) *light shading* – smallest allocation; *intermediate shading* – intermediate allocation; *dark shading* – largest allocation.

subset of computational components (e.g. caches, ALUs, registers, etc.) that together form a complete computing platform. Note that there are an exponential number of unique subsets that can be created from such a set of components, so in principle the number of hardware allocations that can be created from a single computing platform is exponential in the number of computational components or functional units. However, there are two repercussions. The first is that the subsets must themselves form completely functioning computing platforms in order to be candidate hardware allocations. Furthermore, the power gating circuitry required to power on off specific hardware allocations imposes area and energy overheads and therefore the number and types of hardware allocations is limited. Nevertheless, power gating can be used to construct M hardware allocations $\{m_1, ..., m_M\} \in \Phi$, which can in turn be operated at V discrete supply voltages $\{v1, ..., v_V\} \in \Psi$; each hardware allocation-supply voltage pair constitutes a single hardware configuration $c_i, 1 \leq i \leq M \cdot V$. Consequently, each configuration will execute a given task with a unique speed and energy.

An example power gating scheme for hardware allocation creation is shown Figure 1. The figure shows a complete computing platform consisting of 3 instruction fetch units (IFU), 8 execution units (EX), 3 memory management units (MMU), L1 and L2 caches, a phase-locked loop (PLL), voltage regulators (VREG), and a power management unit (PMU). We use power gating to create $M = 3$ hardware allocations, where each shading level corresponds to a single hardware allocation. The lightest shaded components are present in all 3 allocations ($m1$, $m2$, and $m3$) the medium shaded components are present in only the two larger allocations ($m2$ and $m3$), and the darkest shaded components are present in the largest allocation comprising the entire computing platform ($m3$). For example, $m1$ consists of only 4 EX units, $m2$ consists of 6, and $m3$ contains all 8.

Note that the PLL and VREG are used to achieve desired clock speeds and are managed by the PMU, which is also responsible for coordinating the power gating circuitry; therefore, these components must be present in all allocations. Note also that although only one L1 cache and one L2 cache are physically present, the multiple blocks in the figure represent different sizes of the caches, where a portion of the cache is power gated for smaller allocations.

B. Application Profiling

The goal of the application profiling step is to capture an application's execution behavior (e.g. instructions per cycle, cache misses, macro/micro operations, etc.) and quantify how it uses available hardware resources (e.g. ALUs, registers, caches, etc.) as well as how it is affected by its data input parameters (e.g. jpeg, mpeg, audio, etc.). Application profiling provides essential information about an application's achievable performance (instruction-level parallelism) and power consumption (functional unit switching capacitance), enabling key energy and speed trade-offs to be analyzed on both a per-application and a per-hardware allocation basis.

We utilize the cycle-accurate *SimpleScalar* simulator for acquiring per-application profiling statistics for each considered hardware allocation [28]. The recorded statistics include the number of instructions, execution cycles, and functional unit activity. As a result, a unique profiling result is generated for each hardware allocation and can be compared across different hardware allocations at identical time frames by their instruction count id. Note that instruction cycles cannot be used since cache sizes, number of functional units, and bus-width, for example, each potentially impacts the total required execution cycles of an application if the hardware allocation was altered during run time. Thus, we use the instruction id as a synchronization mechanism to compare profiling statistics across various hardware allocations enabled by power gating.

We profile each considered benchmark for each hardware allocation. Profiling at different voltages is not required since voltage scaling primarily impacts the operational speed or clock frequency. Note that more sophisticated profiling must be conducted for an asynchronous processor where the clock rate may vary across functional blocks. However, for our experimental platform we consider a synchronous in-order processor platform that operates at a single global voltage, where all functional units operate at a unified clock rate.

Application profile results can also indicate periods of high application- and hardware-level parallelism, further improving allocation of tasks to available configurations. For example, periods of high instruction throughput can be assigned configurations set at lower voltages, while periods of low throughput can be assigned at higher voltages in order to speed up the execution of the bottleneck while minimally impacting energy consumption. A similar approach is found on conventional modern microprocessors by transitioning to *turbo-mode*, which operates by assigning periods of low thread-level parallelism to higher clock frequencies in order to maximize energy efficiency. Our approach differs since we alter both the hardware platform via power gating and speed via DVS techniques.

The functional unit activity for each profile can then be processed through an event-driven power simulator such as *Wattch* in order to generate energy results for each hardware allocation setting [29]. We extract the energy and timing values at a fine sub-task granularity. The minimum task size (or sub-task) should be sized in accordance to the minimum configuration switching overhead break-even point; for our experiments, we set the minimum sub-task granularity to be 10 instructions. Profiling at a smaller granularity would incur long simulation run times. However, in many cases fine granularity profiling can be performed since it must be done only once for

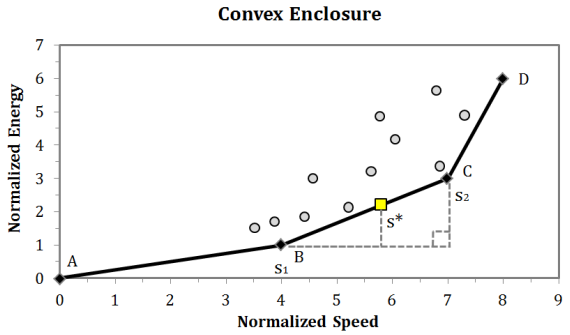


Fig. 2: Convex enclosure comprising configurations A, B, C, and D over all configurations. The virtual speed s^* can be achieved by utilizing speeds s_1 and s_2 under configurations B and C, respectively.

each considered application, input set, and hardware allocation and is done off-line.

C. Single Task Allocation

Dabiri et al. described a technique for using at most two configurations to achieve any *virtual* operating speed [17]. Consider a simple example where there are two configurations c_1 and c_2. To run the system at virtual speed s^* ($s_1 \leq s^* \leq s_2$) for a given interval $[a, b]$, where s_1 and s_2 are the speeds of configurations c_1 and c_2, respectively, for the given task (obtained from profiling), we use configuration c_1 for t_1 seconds and c_2 for t_2 seconds. We calculate t_1 and t_2 as:

$$t_1 = \frac{s_2 - s^*}{s_2 - s_1} \times (b - a), t_2 = \frac{s^* - s_1}{s_2 - s_1} \times (b - a) \quad (1)$$

s^* is the weighted average speed when the system is run at s_1 and s_2, indicating that in the duration of $[a, b]$ the system was operating at the virtual speed of s^*. The key result is that at most two configurations on the *convex enclosure* of the speed-energy trade-off curve of available configurations can be used to obtain any virtual speed of execution using provably minimal energy. The convex enclosure can be defined as the convex piecewise-linear curve connecting points such that all points are on the enclosure or above and to the left of it.

Figure 2 shows an example of a convex enclosure for a task with 15 candidate configurations. *Black diamond* points (A, B, C, and D) represent configurations comprising the convex enclosure. An optimal virtual speed s^* can be achieved by executing the task on at most two configurations on the convex enclosure that bound that virtual speed. Recall that the durations of execution on either configuration can be calculated using Equation 1. The convex enclosure property ensures that no other combination of configurations can execute the task at virtual speed s^* with lower energy. In this example, configuration B would be used for time t_1 and configuration C for time t_2.

Note that because the optimal solution uses at most two configurations, it requires a maximum of only one configuration switch; therefore, the configuration switching overhead is negligible for reasonable task lengths. However, this approach optimizes for only a single task and is optimal only under the assumption that energy consumption is uniform within a task. However, often a computing platform must execute a variety of expected applications, each of which has different degrees of instruction-level parallelism and functional unit usage throughout execution. Our goal is to provide a technical approach to allocating both time and hardware to execute multiple tasks and sub-tasks with minimal energy while maintaining execution deadlines in the presence of both uncertainty and configuration-switching overheads.

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

D. Simulation Environment and Models

We used the *SimpleScalar-ARM* simulator [28] to profile single-threaded ARM7TDMI cycle-accurate application traces for up to 10 million instructions on the hardware allocations shown in Table I. We used *McPAT* [31] to extract dynamic and leakage power values for each allocation. We construct our power model to support five hardware allocations, with each allocation capable of operating at five discrete supply voltages and frequencies (0.6V at 450 MHz, 0.7 at 600MHz, 0.8V at 850 MHz, 0.9V at 1.0 GHz, and 1.1V at 1.15 GHz). We utilize a $Wattch$-based power model to obtain the total energy consumption values for each application at each configuration by using the recorded application profiling results for each functional unit and applying its respective power cost. It is important to note that the total execution time is required for accounting for leakage energy.

E. Configuration Switching Overhead

Switching between configurations has potentially two sources of overhead: 1) power gating overhead caused by switching between hardware allocations (pg); and 2) voltage scaling overhead (dvs). Thus, the total energy overhead $\varepsilon_{i \to j}$ is the sum of the two components $\varepsilon^{pg}_{i \to j}$ and $\varepsilon^{dvs}_{i \to j}$, while the total delay overhead is the maximum between $\delta^{pg}_{i \to j}$ and $\delta^{dvs}_{i \to j}$.

1) Power gating: Power gating has been used as an effective leakage energy saving technique and operates by disconnecting the supply voltage source to the circuit block of interest. We adopt the formulations presented by Hu et al. in accounting for energy and delay transition overheads [21]. We compute the total energy overhead $\varepsilon^{pg}_{i \to j}$ for switching from configuration c_i (with operating voltage v_i and switching capacitance C_i defined by the hardware allocation) to configuration c_j as the amount of energy required to drive the device headers to both power off the blocks present in c_i and power on those in c_j, shown below:

$$\varepsilon^{pg}_{i \to j} = W_H \cdot (C_i \cdot v_i^2 + C_j \cdot v_j^2). \quad (2)$$

where W_H represents the ratio of the total area of the header device to the total area of the power-gated component. As in the work by Hu et al., we use the typically quoted ratio W_H = 0.1 and assume a constant delay overhead of $\delta^{pg}_{i \to j} = 10$ clock cycles for powering up or down functional units.

2) Dynamic voltage scaling: We adopt energy and time overheads when scaling voltages provided by Burd et al. [32], reproduced below:

$$\varepsilon_{i\to j}^{dvs} = (1-\eta)\cdot C_{reg}\cdot \|v_j^2 - v_i^2\| \quad (3)$$

$$\delta_{i\to j}^{dvs} = \frac{2\cdot C_{reg}}{I_{max}}\cdot \|v_j - v_i\| \quad (4)$$

where η and C_{reg} are the voltage regular efficiency and load capacitance, respectively, and I_{max} is the circuit's maximum drive current. We adopt the standard values of $C_{reg} = 10$ μF and $I_{max} = 1$ A to estimate delay and energy overheads roughly on the order of tens of μs and μJ, respectively.

IV. Adaptation: Time Allocation

This section presents an adaptive time allocation algorithm that allocates tasks or sub-tasks to particular hardware configurations for particular amounts of time. In this phase, the goal is to minimize total energy by assigning an execution time and hardware configuration to each task or subtask given a global execution deadline. We address time allocation under the following two scenarios: i) detailed task analysis is not practical, in which case tasks can only be subdivided into coarse-grained sub-tasks or not at all; and ii) accurate profiling of a task is practical, and therefore the task can be split into multiple sub-tasks at a very fine granularity. In the former case, configuration switching overheads (for power gating and voltage scaling) are negligible in the case where task intervals are significantly longer than the switching overheads. However, in the latter, switching overheads impact the final result due to the finer task scheduling granularity and thus optimality can no longer be guaranteed.

A. Coarse-Grained Profiling

Assumptions. N tasks (or sub-tasks) have been profiled for each supported hardware configuration; resulting speed-energy points for each task on each configuration have no uncertainty; each task t_i can be run on C_i configurations $c_{i,1}$, $c_{i,2}$, ..., c_{i,C_i} on its convex enclosure with speeds $s_{i,1}$, $s_{i,2}$, ..., s_{i,C_i}, respectively; the tasks are of sufficient length that configuration switching overheads are negligible; and an overall speed requirement s_{global} is expected.

Problem Formulation. Our key objective is to determine the set of configurations with which to process a given set of tasks such that the total energy consumption of the system is minimized, while completing within a specified deadline. Recall that for a single task t_i, a target speed s_i^* can be achieved optimally in terms of energy using the method described in Section III-C. Therefore, this problem can be reduced to finding the set of speed requirements s_i^* for each task t_i such that the global speed requirement s_{global} is met with minimal overall energy consumption.

We solve this problem using convex or piecewise linear programming. Each piecewise-linear segment corresponds to the line segment connecting two consecutive configurations on the convex enclosure of a task. The energy consumed by each task can be represented by the equations below:

$$e_i^*(s_i^*) = \begin{cases} f_{i,1}(s_i^*) & s_{i,1} < s_i^* \le s_{i,2} \\ f_{i,2}(s_i^*) & s_{i,2} < s_i^* \le s_{i,3} \\ \ldots & \\ f_{i,C_i-1}(s_i^*) & s_{i,C_i-1} < s_i^* \le s_{i,C_i} \end{cases} \quad (5)$$

The energy consumed per task $e_i^*(s_i^*)$ is a function of the energy-speed slope $f_{i,j}(s_i^*)$ between the speed interval ($s_{i,j}$, $s_{i,j+1}$] that includes the virtual speed s_i^*. Since the energy-speed slope for each task is a linear function of speed, we can determine the ratio of time spent, $x_{i,j}$ for task t_i under configuration c_j. The objective is to minimize the total energy for all tasks:

$$\min \sum_{i=1}^{N} e_i^*(s_i^*) \quad (6)$$

subject to the constraints:

$$\sum_{j=1}^{C_i} s_{i,j}\cdot x_{i,j} = s_i^* \quad \forall i \quad (7)$$

$$\sum_{j=1}^{C_i} x_{i,j} = 1 \quad \forall i \quad (8)$$

$$0 \le x_{i,j} \le 1 \quad \forall i,j \quad (9)$$

$$\sum_{i=1}^{N} \frac{1}{s_i^*} \le \frac{1}{s_{global}^*} \quad (10)$$

Equations 7-9 specify that the achieved virtual speed s_i^* for each task may only be achieved by utilizing real speeds from actual configurations for a total of 100% of the allotted time. Equation 10 provides the guarantee that the time allotted for each task must satisfy the global execution time deadline. Therefore, the convex piecewise-linear formulation determines the optimal virtual speed for each task (s_i^*) such that the global speed deadline is met and the total energy for all tasks is minimized. It is important to note that because the formulation is convex, the optimal solution will include at most 2 consecutive configurations for each task.

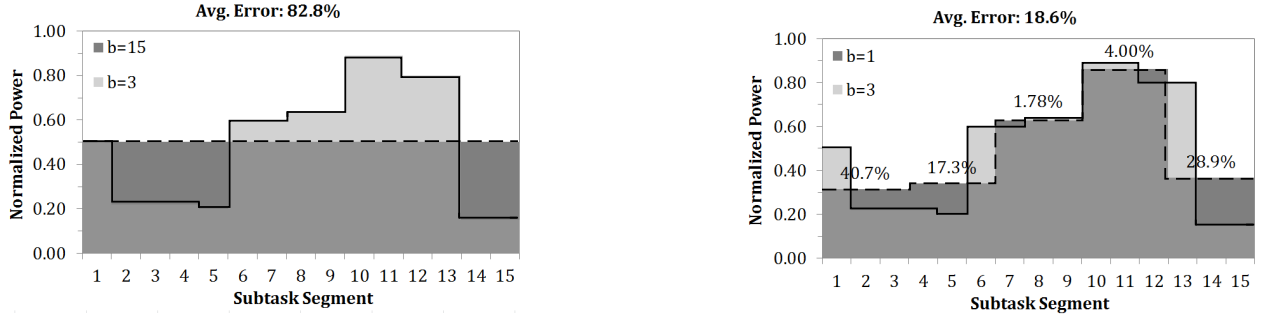
Furthermore, the solution is globally optimal under the following assumptions:

(i) *Tasks are long enough that energy and delay for switching configurations is negligible.* The formulation does not consider energy and delay overheads for configuration switching. The key observation is that because there is only at most one configuration switch for each task, and at most one configuration switch between tasks, the time and energy spent in context switching is negligible compared to the overall time and energy expenditure.

(ii) *Energy consumption is constant throughout the execution of a task.* The formulation does not differentiate between which portions of a task are executed under which configuration. It simply determines the ratios of time, $x_{i,j1}$ and $x_{i,j2}$, for which to execute task i under configurations $c_{i,j1}$ and $c_{i,j2}$, respectively; the actual sections of the task that are run under each configuration are arbitrary. However, energy consumption within a task is highly variable, as we discuss in the next subsection. Therefore, we use this formulation only when application profiling is coarse-grained (i.e. tasks are split into relatively long sub-tasks) and thus the energy consumption distribution within a subtask is unknown.

B. Fine-Grained Profiling

Assumptions. N tasks (or subtasks) have been profiled for each supported hardware configuration; resulting speed-



(a) (b)

Fig. 3: Non-uniform task power consumption (solid lines) and average power consumption (dashed lines) for $jpegdec$ application under two sub-task block sizes: b=15 (a) and b=3 (b). Displayed percentages denote the error from assuming a uniform power consumption within a block. Subdividing a task into smaller sub-tasks enables greater accuracy as shown by reduced error achieved by block size $b = 3$ (18.6% overall error) vs. the error achieved with $b = 15$ (82.8% overall error).

energy points for each task on each configuration have no uncertainty; each task t_i can be run on C_i configurations $c_{i,1}$, $c_{i,2}$, ..., c_{i,C_i} on its convex enclosure with speeds $s_{i,1}$, $s_{i,2}$, ..., s_{i,C_i}, respectively; configuration switching delay and energy overheads are $\varepsilon_{j\to k}$ and $\delta_{j\to k}$, respectively, for switching from configuration $c_{i,j}$ to $c_{i,k}$ for any task t_i; and an overall speed requirement s_{global} is expected.

Problem Formulation. Our objective is to allocate a single configuration to each task such that total energy consumption of the system is minimized while a specified global execution deadline is satisfied. In this case, because configuration switching overheads are potentially significant (i.e. tasks are of relatively small length), each task is run on only a single configuration and thus the virtual speed of a task is equivalent to the actual speed of the configuration.

Figures 3a and 3b show the actual non-uniform power consumption of a task, $jpegdec$ (solid lines). In Figure 3a, the task is profiled at a block size of $b = 15$ instructions, with uniform power consumption assumed within the block, leading to an average error of 82.8%. In Figure 3b, a much smaller block size $b = 3$ is used for fine-grained profiling, leading to a much smaller average error of only 18.6%. This error ultimately translates to suboptimal configuration assignment to subtasks, and therefore minimizing it should be a crucial goal. Therefore, tasks should be profiled at as small a granularity as is practical to minimize energy.

However, recall that at this granularity, configuration switching overheads become significant. To solve the configuration allocation problem under these conditions, we propose another dynamic programming approach, summarized in Figure 4. We construct a directed acyclic graph (DAG), where each node represents the energy and time expenditure for running a block of instructions (or subtask) $b_i, 1 \leq i \leq M$ on configuration $c_j, 1 \leq j \leq C$. A directed edge between two nodes represents the energy and time overheads for switching from the first configuration to the next. In the figure, each vertical stage corresponds to a single subtask or block, and each horizontal set of nodes corresponds to a single configuration, with edges from all configurations at one block to all configurations at the next block. There is also a single start node and a single end node.

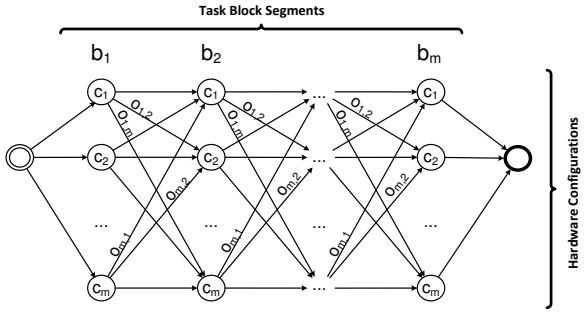


Fig. 4: Directed acyclic graph (*DAG*) for obtaining the shortest path when considering switching overheads, as formulated by our dynamic programming approach. Each node ($c1$, $c2$, ..., c_m) per block column represents a given configuration setting (m possible configurations). Energy and delay overheads are modeled with $O_{i,j}$ where i, where $i \neq j$.

The minimal energy set of configurations on which to execute each subtask with global speed requirement s_{global}, then, corresponds to the shortest path from the start node to the end node in terms of energy while not exceeding the global time constraint. This can be solved using the standard dynamic programming approach, with the following caveat. In order to prevent combinatorial explosion of paths that need to be maintained (in order to meet the speed requirement), a minimal amount of bookkeeping must be performed. This bookkeeping maintains at each node only a maximum number of representative Pareto optimal points in terms of speed and energy. In other words, we conduct uniform sampling of Pareto optimal speed-energy points at each node.

V. Experimental Results

In this section, we present the experimental results for our approaches using 12 standard benchmarks and 26 hardware configurations, composed of 5 base hardware allocations, 5 possible operating voltage, and one configuration representing the situation where all hardware is power gated off [34].

We evaluate the approaches proposed in Section IV, where a single task or set of tasks (or subtasks) are assigned hardware configurations on which to execute to minimize energy in

the presence of a global speed requirement. Specifically, we evaluate the energy consumption using the following four methods: a) DVS alone, using the best single configuration that satisfies the deadline; b) DVS and power gating under the assumption of uniform energy consumption across the entire task; c) DVS and power gating after coarse-grained task profiling to reduce the error in energy estimation; and d) DVS and power gating after fine-grained task profiling to further minimize energy.

We use the first approach as a baseline for comparison, where only a single hardware allocation under multiple voltage settings is supported by the hardware platform. The second approach is the one proposed by Dabiri et al., where at most two configurations are used throughout the execution of the task. Finally, the third and fourth are evaluations of our approaches presented in Sections IV-A and IV-B, respectively. For coarse-grained and fine-grained profiling, we profiled the benchmarks using block sizes of 100,000 and 10,000 instructions, respectively.

Note that in order to do a fair energy comparison, we evaluate all approaches under the same hard performance constraint, which was set to mid-range achievable clock frequency of 800MHz. Figure 5 presents the energy consumption for each of the four considered configuration selection methods. The results are normalized to the energy consumed by the best single configuration (*1-conf.*) that satisfies the desired deadline or performance constraint. We compared the *1-conf.* result against the best 2-configuration result (*2-conf*), our convex programming approach under coarse-grained profiling (cv), and our dynamic programming-based approach under fine-grained profiling (dp). We again evaluated the approaches were evaluated under three task set assumptions: 1) per application (single task); 2) separate encode ($*enc$ - 6 tasks) and decode ($*dec$ – 6 tasks) classes; and 3) an aggregate task set comprised of all benchmarks ($*all$ - 12 tasks).

The results in Figure 5 demonstrate the additional benefits of assigning configurations at smaller subtask granularities. Our results indicate that increasing the number of configuration selection opportunities, achieved by decreasing the subtask size, is shown to achieve greater savings. The number of configuration selection opportunities is 1, 2, 10, and 100 for *1-conf*, *2-conf*, *cv*, and dp, respectively. The best configuration that can be selected for the *1-conf* is for the case where the specified timing constraint falls exactly on the same energy and speed point of a base configuration. A *1-conf.* solution would yield the same result as *2-conf* since the virtual speed derived using only one configuration for the entire task would be used. As a result, the additional savings achieved by a *2-conf.* method over the *1-conf.* is limited by the difference between the two surrounding configurations on the convex enclosure. Our results show that utilizing *2-conf.* achieves a *3.39X* max (*1.46X* avg.) savings over the best individual configuration *1-conf*.

Additional energy savings is achieved by subdividing the entire task into smaller subtask sizes, and is shown by the convex programming solution *cv*. For the *cv* results, the entire task (10M instructions) was subdivided into equal 100K instructions, grouped into subtasks. The convex programming solution determines the optimal time allocation for each subtask. As a result, the cv method is guaranteed to obtain a solution that achieves greater savings than *2-conf.* by leveraging the non-uniform energy and speed behavior across the subdivided subtasks, achieving up to *4.64X* max (*2.44X* avg.) energy savings over *1-conf.* Therefore, a convex programming solution capable of assigning configurations at smaller subtask granularities will always yield a superior solution. Recall that in this case, configuration switching overheads are negligible due to long tasks size with respect to the switching granularity (assuming the convex enclosure for each subtask is different).

Eventually the overhead transitions incurred by configuration switching at smaller subtasks reach a limit where the energy and speed costs become comparable to costs associated in running the task under a constant configuration. Therefore, in order to maximize energy savings for these scenarios, it becomes even more critical that configurations are selected in such a way that energy is minimized without compromising performance targets. For our experiments, subdividing tasks into equal 100 subtasks (e.g., 1K instructions = $\frac{1M}{100}$) results in comparable consumed energy and latency requirements to the associated energy (6-20 μJ) and timing transition costs (μs or approx. 600-3000 clock cycles) when performing DVS. Power gating overheads were negligible at these subtask granularities. Therefore, a convex programming solution capable of assigning configurations at a subtask granularity of 100 subtasks would produce a solution that will potentially violate timing constraints. The convex programming solutions are limited since there is no notion in the formulation of the configuration of the previous or subsequent tasks. Although this solution is provably optimal, it is so only in the case where overheads are negligible.

A dynamic programming dp solution becomes a natural choice for configuration selection at ultra-fine subtask granularities. Our results show that it consistently generated the minimal energy configuration among the considered methods, achieving up to 8.27X max (4.11X avg.) energy savings while satisfying timing constraints. The dynamic programming solution achieves the best solution due to its advantage of assigning configurations at smaller granularities, effectively leveraging energy and speed trade-offs across all subtasks of a given application or application set, while accounting for transition overheads.

VI. Conclusion

We have presented a framework that addresses energy minimization while accounting for performance constraints in the context of determining hardware configurations and task time allocation. Given a set of operating configurations and a set application behavior, we have presented algorithms for energy optimization under the following two scenarios: 1) application profiling is done at a coarse granularity and therefore tasks are long enough to render configuration switching overheads negligible (convex enclosure and convex programming); and 2) application profiling is done at a fine granularity, where configuration switching overheads are significant (dynamic programming). Finally, we have evaluated our algorithms and conducted quantitative comparison with previous approaches across 15 sets of 12 real applications, achieving up to 4.64X (2.44X average) energy improvements.

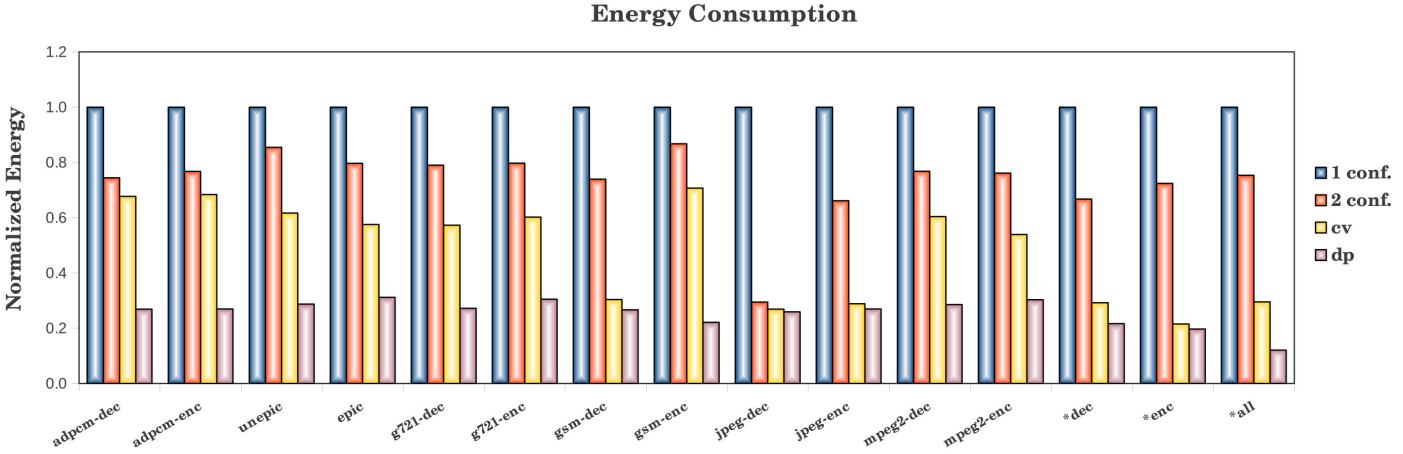


Fig. 5: Energy consumption for each task and task set *{*enc, dec, all*} under various energy reduction methods: 1) best single configuration (*1 conf.*); 2) best two configurations employing DVS and power gating (*2 conf.*); 3) convex optimization with coarse-grained profiling (*cv*); and 4) dynamic programming with fine-grained profiling (*dp*).

ACKNOWLEDGEMENT

This work was supported in part by Samsung under award GRO-20130123 and by NSF under Award CCF-0926127.

REFERENCES

[1] Advanced Configuration and Power Interface (ACPI), http://www.acpi.info, 2012.

[2] J. S. Chase et al., "Managing energy and server resources in hosting centers centers," *SIGOPS*, pp. 103-116, 2001.

[3] F. Yao et al., "A scheduling model for reduced CPU energy," *FOCS*, pp. 374-382, 1995.

[4] T. Ishihara and H. Yasuura, "Voltage scheduling problems for dynamically variable voltage processors," *ISLPED*, pp. 197-202, 1998.

[5] J.-J. Chen et al., "1+ϵ approximation clock rate assignment for periodic real-time tasks on a voltage-scaling processor," *EMSOFT*, pp. 247-250, 2005.

[6] S. Irani et al., "Algorithms for power savings," *SODA*, pp. 37-46, 2003.

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

[8] Y. Lee et al., "Scheduling techniques for reducing leakage power in hard real-time systems", *ECRTS*, pp. 105-112, 2003.

[9] L. Niu and G. Quan, "Reducing both dynamic and leakage energy consumption for hard real-time systems," *CASES*, pp. 140-148, 2004.

[10] R. Jejurikar and R. K. Gupta, "Procrastination scheduling in fixed priority real-time systems," *LCTES*, pp. 57-66, 2004.

[11] A. P. Chandrakasan et al., "Optimizing power using transformations," *TCAD*, pp. 12–31, 1995.

[12] N. A. Conos et al., "Maximizing Yield in Near-Threshold Computing under the Presence of Process Variation," *PATMOS*, pp. 1-8, 2013.

[13] G. Qu and M. Potkonjak, "Energy minimization with guaranteed quality of service," *ISLPED*, pp. 43–48, 2000.

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

[15] D. Kirovski and M. Potkonjak, "System-level synthesis of low-power hard real-time systems," *DAC*, pp. 697-702, 1997.

[16] H.-R. Hsu et al., "Multiprocessor synthesis for periodic hard real-time tasks under a given energy constraint," *DATE*, pp. 1–6, 2006.

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

[18] N. A. Conos et al., "Provably minimal energy using coordinated DVS and power gating," *DATE*, pp. 1–6, 2014.

[19] S. Mutoh et al., "I-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS," *JSSC*, vol. 30, no. 8, pp. 847-853, 1993.

[20] J. Kao and A. Chandrakasan, "Dual-threshold voltage techniques for low-power digital circuits," *JSSC*, vol, 35, no. 7, pp. 1009-1018, 2000.

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

[22] S. Rele et al., "Optimizing static power dissipation by functional units in superscalar processors," *CC*, pp. 261-275, 2002.

[23] N. Madan et al., "Guarded power gating in a multi-core setting" *ISCA*, pp. 198-210, 2010.

[24] H. Xu et al., "Stretching the limit of microarchitectural level leakage control with adaptive light-weight Vth hopping," *ICCAD*, pp. 632-636, 2010.

[25] P.-H. Wang et al., "Power gating strategies on GPUs," *TACO*, vol. 8, no. 3, pp. 1-25, 2011

[26] I. Hong et al., "Synthesis techniques for low-power hard real-time systems on variable voltage processors," *RTSS*, pp.178–187, 1998.

[27] J. Flinn and M. Satyanarayanan, "Energy-aware adaptation for mobile applications," *SIGOPS*, pp. 48-63, 1999.

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

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

[30] I. Hong et al., "Power optimization of variable-voltage core-based systems," *TCAD*, pp. 1702–1714, 1999.

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

[32] T. D. Burd and R. W. Brodersen, "Design issues for dynamic voltage scaling," *ISLPED*, pp. 9–15, 2000.

[33] I. Hong et al., "On-line scheduling of hard real-time tasks on variable voltage processor." *ICCAD*, pp. 653–656, 1998.

[34] C. Lee et al., "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems," *MICRO*, pp. 330–335, 1997.

[35] S. Wei et al., "Aging-based leakage energy reduction in FPGAs,"*FPL*, pp. 1–4, 2013.