Journal of
Low Power Electronics
Vol. 5, 1–15, 2009

Energy-Aware Compilation for Embedded Processors with Technology Scaling Considerations

Po-Kuan Huang and Soheil Ghiasi*

Department of Electrical and Computer Engineering, University of California, Davis, CA 95616, USA

(Received: xx Xxxx Xxxx; Accepted: xx Xxxx Xxxx)

With scaling of technology feature sizes, the share of leakage in total energy consumption of digital systems is on the rise. Conventional dynamic voltage scaling (DVS) techniques fail to accurately address the impact of scaling on system energy consumption breakdown, and hence, are incapable of achieving energy efficient solutions in all technology nodes. To overcome this problem, we propose utilizing adaptive body biasing (ABB) to adjust transistors' threshold voltage at runtime. While ABB has intrinsic limitations with deep sub-micron scaling, we demonstrate that it can be favorably combined with DVS to reduce overall energy consumption down to 45 nm technology node. We develop a leakage-aware compilation methodology for embedded applications under hard or soft timing constraint. Our technique targets embedded processors with both DVS and ABB capabilities, and has the unique advantage of jointly optimizing active and leakage energy dissipation. Considering the delay and energy overhead of switching between operating modes of the processor and execution deadline constraints, our compiler improves the energy consumption of the generated code by average of 21.66% at 90 nm. While our technique's improvement in energy dissipation over conventional DVS is small (6.43%) at 130 nm, the average improvement continues to grow to 12.23%, 18.63% and 22.16% for 90 nm, 65 nm and 45 technology nodes, respectively. Extensive experiments validate the effectiveness of our approach, explore the involved trade-offs, and offer insights into future trends with respect to technology scaling.

Keywords: Embedded and Realtime Systems, Compilation, Leakage, Energy Optimization, Dynamic Voltage Scaling.

1. INTRODUCTION

Due to its significant impact on battery life, system density, cooling costs and reliable operation, energy consumption has become one of the most important design concerns for digital systems. In previous technology nodes, active power was the primary contributor to total power dissipation of a CMOS design. Quadratic dependence of active power on supply voltage, along with the lower order impact of supply voltage on clock frequency motivated the idea of frequency and supply voltage scaling for processors.[35] In this scheme, supply voltage and hence, operating frequency of processors are reduced to save energy whenever full performance is not required. However, the leakage power, whose share in total power increases with the scaling of CMOS technology, is not explicitly addressed using this technique. Consequently, the effectiveness of traditional voltage and frequency scaling is limited with advancement of technology.[10] Figure 1(a) illustrates typical share of active and leakage power consumption in total system power across several technology nodes.

Adaptive body biasing (ABB) is a well-known CMOS design technique that allows runtime adjustment of transistors' threshold voltage. Threshold voltage affects both leakage and delay of the transistors. Hence, its effect can be combined with supply voltage scaling to minimize *overall* power consumption for a given frequency.[33] Unlike traditional dynamic voltage scaling (DVS), combined dynamic voltage scaling and adaptive body biasing (DVS+ABB) has the potential of holistically optimizing system power consumption by considering both active and leakage power. Exploiting the energy savings potentials of this technique, greatly depends on *mode switching* policies that dynamically adjust both supply and body bias voltage of the processor.

*Author to whom correspondence should be addressed.
Email: ghiasi@ucdavis.edu

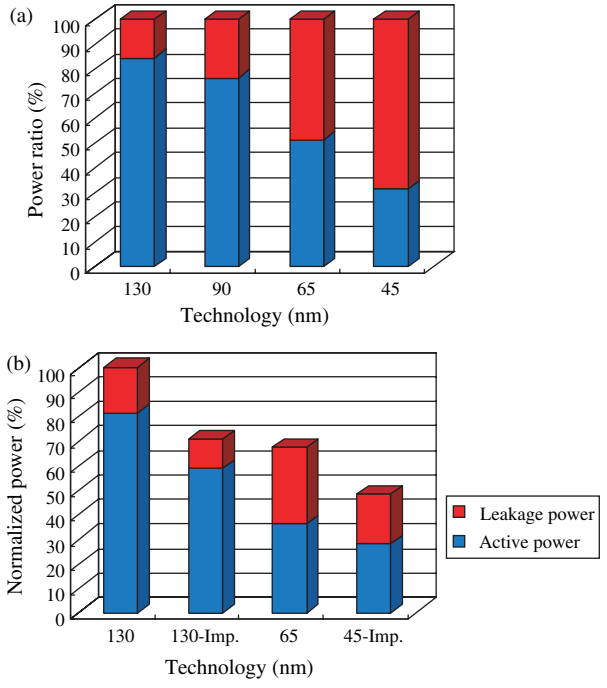

Fig. 1. (a) Normalized share of active and leakage in overall power for typical designs across the technologies. (b) Combined DVS and ABB optimizes both leakage and active power.

Figure 1(b) shows the breakdown of energy savings at 130 nm and 65 nm technologies by utilizing both DVS and ABB under deadline constraint. The labels with "-Imp" postfix refer to energy improvement with DVS+ABB. In this figure, the power consumption of a typical processor, assumed to have six million logic gates and peak operating frequency of 600 MHz with 10% acceptable performance loss, is estimated before and after application of DVS+ABB (Section 3). As the figure suggests, combined DVS and ABB decreases both active and leakage energy. More importantly with advance of technology, it optimizes leakage component of overall power consumption more aggressively.

Although ABB is known to have limitations with aggressive scaling, its effectiveness is demonstrated in feature sizes as small as 50 nm.[28] In Section 3, we review the background information and discuss diminishing energy return of ABB with scaling. Based on transistor models derived from Berkeley predictive technology model,[1] we argue that ABB remains beneficial in 45 nm technology node. Subsequently, we restrict ourselves to study the impact of ABB+DVS in technology nodes that are not beyond 45 nm.

In this paper, we present a compilation methodology that targets embedded processors with joint DVS and ABB capabilities. The idea is to statically analyze the application and integrate mode switching policy into the generated code by inserting mode switch instructions. By judiciously placing mode switch instructions between selected basic blocks of the code, our compiler generates code that is optimized for overall energy consumption.

We target embedded realtime applications that demand responsiveness. We formulate static frequency assignment problem as an MILP instance. The parameters and constants of MILP are derived by profiling the application using the input data associated with worst-case execution time (WCET). As a result, frequency assignment is directly dependent on WCET input. Our approach can readily handle soft realtime applications, when worst-case profiling is replaced with average-case analysis.

Switching between processor modes can also be dynamically performed by the operating system. However, many embedded systems have constrained hardware resources, and hence, demand a light-weight operating system. Static integration of frequency scaling into the code mainly serves such systems, because it frees the operating system from having to perform periodic frequency scaling. As we will show later, static integration of frequency scaling policies into the generated code incurs negligible cost in terms of code size, and energy and delay overhead.

In addition, we argue that compilers can obtain information about the code, e.g., program taking a particular execution path or waiting on a definite cache miss, that the operating system cannot access. As a result, compilers can exploit a different type of execution slack, which would not be utilized by operating systems. Utilizing this type of slack, it has been shown that collaborative voltage scaling, enabled by synergy between compiler and operating systems, is superior to conventional OS-enabled voltage scaling schemes.[26]

We have integrated our static analysis and static frequency scaling algorithms into SUIF compilation framework. We generate executable code for a number of embedded applications using the developed compiler. We report extensive experimental results to demonstrate effectiveness and tradeoffs of using our method. Compared to baseline compilation, our compiler improves the energy consumption of the processor by 14.45%, 21.66%, 32.65% and 40.64% for 130 nm, 90 nm, 65 nm and 45 nm technologies, respectively. Compared to traditional DVS-only optimization, we improve the average energy consumption by 6.43%, 12.23%, 18.63% and 22.16% for the four aforementioned technologies.

2. RELATED WORK

Many different circuit techniques have been proposed to reduce energy dissipation of a digital system. Dynamic voltage scaling (DVS) and adaptive body biasing (ABB) are two such techniques that have been widely used in designs. Extensive research has been carried out to minimize dynamic power consumption of a CMOS design. In microprocessor-based systems, DVS is used to regulate the supply voltage and the clock frequency under timing

constraints. DVS has been utilized in several fabricated academic and commercial processors. In Ref. [35], an ARM V4 processor is implemented and demonstrated to be energy efficient. Intel XScale[2] and Transmeta LongRun2 technology[3] offer commercial embedded processors that utilize DVS.

On the other hand, ABB has been previously used to adjust the threshold voltage of the transistors and reduce the leakage energy of the design. Various techniques for implementing ABB, and the associated leakage power reduction have been discussed in the areas of circuit design and optimization, and semiconductor process technology. Kao et al. present a combined DVS and ABB architecture to achieve the lowest active power consumption.[19] The real-time ABB techniques are also proved effective to reduce the leakage power consumption.[11] Researchers have shown that an optimum reverse body bias voltage can be used to minimize the standby leakage power consumption.[5] The idea is implemented in Intel 80200 processor, designed based on XScale,[8] to reduce standby leakage.[9] In contrast, we utilize body biasing adaptively in active mode to reduce energy during application execution.

DVS regulates the supply voltage of the system and hence, reduces both dynamic and leakage power consumption of a design. The lower supply voltage positively impacts the leakage as well as the dynamic energy. Nevertheless, energy savings can be improved by coordinated DVS and ABB.[33] As an example, Yan et al. proposed an algorithm for execution of a task graph with real-time constraints on DVS+ABB enabled distributed embedded systems.[37]

Software techniques for judicious voltage and frequency scaling are crucial in delivering low power solutions using the underlying circuits. In order to reduce the energy consumption of low power processors running realtime workload, a controller needs to throttle the operating frequency with respect to deadline constraints. There has been extensive research on dynamic voltage scaling algorithms and operating systems. The majority of existing techniques, however, target only DVS-enabled processors and only consider frequency regulation using the operating system (inter-task level). Such techniques adjust the frequency between task boundaries, and cannot exploit intra-task slack to throttle supply voltage.

Many task-level voltage scheduling algorithms have been proposed for such processors. In Ref. [30], for example, a DVS technique is integrated into the scheduler and task manager of the operating system and its efficiency is demonstrated. Kim et al. propose an on-line slack estimation heuristic and an associated task-level voltage scheduling algorithm.[36]

Several research efforts have proposed methods to perform voltage scaling using intra-task execution information.[4, 7, 13, 26] An analytical study of potential power savings using intra-program DVS is reported in Ref. [13]. In Ref. [4], checkpoints indicating the voltage scaling points are inserted to program during compilation. Hsu et al. introduce an algorithm that identifies the program regions with time slack for processor, and implement it as a source-to-source transformation.[7] In another interesting work, compiler and operating system level optimization are coordinated. Compiler instruments the code to inform the operating system with the number of cycles required to meet the application deadline at different point of its execution.[26]

None of the aforementioned software techniques consider leakage power, and the effect of technology scaling on the validity of their results. However, with the continuing shrinkage of the device sizes, techniques that only target dynamic power will be less accurate and inefficient.[10] Previously, we developed a fast heuristic algorithm for compiling large programs onto a DVS+ABB enabled processor.[17, 18] We also reported preliminary results for a MILP-based technique for intra-program supply and substrate voltage throttling.[16] In this paper, we extend[16] with elaborations on hardware model, scaling considerations, and MILP-specific discussions such as the tradeoff between solver runtime and energy savings.

3. CIRCUIT POWER AND PERFORMANCE UNDER DVS AND ABB

3.1. Background

In this section, we briefly overview the impact of supply voltage and body bias on processor's frequency and power consumption. Referring to related literature, we derive threshold voltage, power consumption, and the performance of the design as functions of its supply and bias voltages.[27, 28, 33] Subsequently, we proceed to present the power and performance parameters of the proposed embedded processor with DVS and ABB capabilities.

DVS allows the microprocessor to scale its supply voltage and operating frequency at runtime. With semi-linear reduction in supply voltage and frequency, DVS obtains quadratic saving in dynamic energy. In practice, the processors often have several distinct voltage–frequency modes. Modes with higher frequency are implemented with higher supply voltage, in which, the processor runs faster and consumes more energy for executing a particular task. DVS has proved to be effective in reducing system dynamic energy consumption, and is being used in several commercial processors such as Intel XScale,[2] and Transmeta LongRun2 technology.[3]

ABB controls the voltage level of the chip substrate using a voltage regulator and hence, adjusts the threshold voltage of all transistors simultaneously. As the reverse body bias (RBB) voltage is applied to the chip, the threshold voltage of transistors is increased and their subthreshold leakage current is reduced. The voltage regulators can be controlled via software instructions.[35] The bias voltage

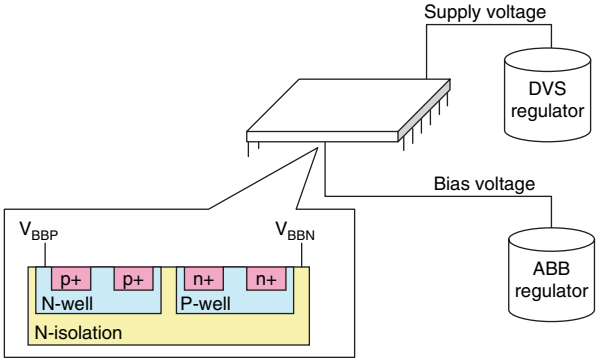

Fig. 2. Sketch of adaptive body biasing implementation.

for all of transistors on the chip can be set by adjusting the voltage V_{BBN} of the NMOS and V_{BBP} of the PMOS.

Our work concerns energy-aware compilation for ABB+DVS enabled processors, and does not deal with their design and implementation. We should point out, however, that ABB is simple to implement. Figure 2 shows a rough sketch of ABB implementation for a die. The area overhead to support ABB is quite small, assuming an off-chip voltage regulator. ABB supply transistors can be provided in the pad ring only, occupying otherwise empty space within the supply pins. Moreover, the connections to substrate are routed sparsely because they only provide low current to change the substrate bias. Thus, their impact on logic and routing density is minimal.[9]

Although the reverse body bias technique is proved to be effective in both 70 nm and 65 nm technologies,[27, 33] recent research points out that there are limitations with the scalability of reverse body biasing technique.[21, 32] In the nano-scale technology, RBB can significantly increase drain induced barrier lowering (DIBL) and band-to-band tunneling (BTBT) leakage currents from source or drain to highly-doped substrate. Increased BTBT leakage currents might eliminate the power-saving benefits from RBB.[27, 28]

Figure 3 illustrates the scalability issue of RBB. The trends are calculated using Berkeley predictive technology models for 45 nm technology. The figure shows the breakdown of normalized leakage power. With increase

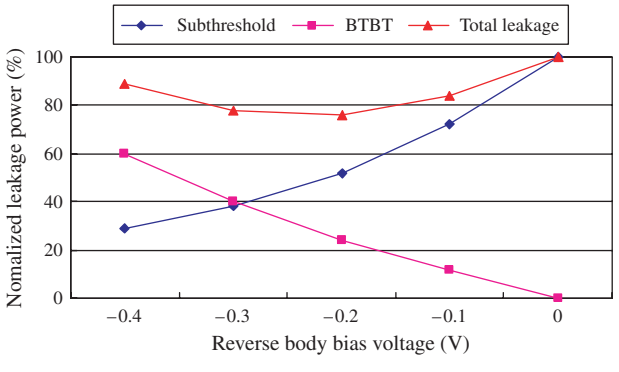

Fig. 3. Normalized leakage power breakdown in the 45 nm model.

of reverse body bias, the "subthreshold leakage" component of total leakage is reduced. However, the "BTBT leakage" current increases exponentially with reduction of RBB voltage. It follows that an optimal bias voltage selection method has to take into account both subthreshold leakage current and BTBT leakage current. In our depicted example, the overall leakage power is minimal around −0.2 body bias voltage.

The increase in BTBT current limits the overall savings that one can expect from RBB. The share of BTBT in total leakage increases with scaling, and hence, potential gain of RBB is curtailed as technology scales. Previous research has shown that by careful selection of an optimal RBB voltage which minimizes the overall energy consumption, RBB can still achieve energy saving in 50 nm technology.[28] Consequently, we consider both subthreshold leakage and BTBT leakage currents to investigate the scalability of DVS+ABB optimization in technologies up to 45 nm (Fig. 3).

3.2. Threshold Voltage

The threshold voltage of a short-channel MOSFET transistor is given by:

$$V_{th} = V_{th_0} + \gamma(\sqrt{\phi_S - V_{bs}} - \sqrt{\phi_S}) + \theta_{DIBL} V_{dd} + \Delta V_{NW} \quad (1)$$

where V_{th_0} is the threshold voltage at zero bias voltage, γ, ϕ_S, and θ_{DIBL} are constants for a given technology, V_{bs} is the body bias voltage between the substrate and source of the transistor, ΔV_{NW} is a constant that models narrow width effects, and V_{dd} is the supply voltage.[29] If $|V_{bs}| \approx \phi_S$, the threshold voltage can be linearized:

$$V_{th} = V_{th_1} - K_1 \cdot V_{dd} - K_2 \cdot V_{bs} \quad (2)$$

where K_1, K_2, and V_{th_1} are constants.[33] Equation (2) formalizes the impact of supply and body voltage scaling on the threshold voltage.

3.3. Power Consumption

The power consumption of CMOS circuits includes active, leakage, and short circuit power. The short circuit power consumption is much smaller than active and leakage power, and is negligible. The dynamic power P_d is given by:

$$P_d = C_{eff} V_{dd}^2 f \quad (3)$$

where C_{eff} is the average switched capacitance per cycle, and f is the clock frequency. The static (leakage) power consumption P_s can be approximated as:

$$P_s \approx V_{dd} I_{sub_n} + |V_{bs}| I_{BTBT} \quad (4)$$

where I_{sub_n} is the subthreshold leakage current, and I_{BTBT} is the band-to-band junction tunneling leakage currents in

the NMOS device.[24, 25, 28, 31] Subthreshold leakage current and I_{BTBT} can be modeled as:

$$I_{\text{sub}_n} = K_3 e^{K_4 V_{\text{dd}}} e^{K_5 V_{\text{bs}}} I_{\text{BTBT}} \approx J_1 e^{-J_2 V_{\text{bs}}} \quad (5)$$

where K_i's and J_i's are technology dependent constants.[27, 28, 33] Considering the relation between subthreshold leakage current, operating conditions (frequency, supply voltage, and body bias) and process technology, the overall power consumption can be summarized as:

$$P = C_{\text{eff}} V_{\text{dd}}^2 f + V_{\text{dd}} K_3 e^{k_4 V_{\text{dd}}} e^{K_5 V_{\text{bs}}} + J_1 |V_{\text{bs}}| e^{-J_2 V_{\text{bs}}} \quad (6)$$

3.4. Delay

The frequency of a processor is determined by the delay of its critical path, which in turn, is determined by the delay of the constituting gates. Both the power supply and the threshold voltage of the constituting transistors impact the gate delay. The delay of complex gates remains proportional to the delay of a standard inverter. As a result, the path delay can be modeled similarly to the alpha-power model of an inverter:

$$t_{\text{inverter}} = \frac{L_d K_6}{(V_{\text{dd}} - V_{\text{th}})^{\alpha}} \quad (7)$$

where L_d is the logic depth of the path,[20, 31] K_6 is constant for a given process, and α is a measure of velocity saturation. Therefore, the frequency of the processor as a function of its technology and operating conditions can be approximated as:

$$f = (L_d K_6)^{-1}((1 + K_1)V_{\text{dd}} + K_2 V_{\text{bs}} - V_{\text{th}_1})^{\alpha} \quad (8)$$

Equation (8) represents clock frequency as a function of both supply and body bias voltages. If we have no control over V_{bs}, as is the case with traditional voltage scaling, there is a unique minimum supply voltage (energy optimal) that would force the processor to operate at a given frequency. However, having DVS+ABB capability allows many potential settings of supply and body bias to force the processor to operate at a given frequency. For a given target clock frequency, there is a unique "energy optimal" pair of supply and body bias voltages that leads to minimum energy consumption for running the processor at that frequency.

4. DVS+ABB ENABLED PROCESSOR MODEL

4.1. Operating Modes

We target a single-issue processor that can operate at several discrete frequencies. According to Section 3, each frequency is associated with a corresponding pair of supply and body bias voltages that allow operation of the processor at that frequency. The combination of the three parameters, i.e., frequency, supply voltage and body bias, constitute an *operating mode* of the processor. The processor is assumed to be able to switch between operating modes by execution of a specialized instruction, referred to as *mode switch instruction*. Given an operating mode, a mode switch instruction can set both the supply voltage and body bias of the processors to switch to that operating mode. Note that the frequency is a function of supply and body bias voltage, and does not need to be specified separately.

Execution of the mode switch instruction, or equivalently switching between modes, incurs delay and energy penalty. Both delay and energy penalty depend on voltage difference of the two modes involved in switching. The deadline requirement and energy optimization will be handled by our compilation methodology, and hence, the architecture does not have to utilize a power and deadline monitoring mechanism.

According to Eq. (8), there are infinitely many (supply voltage, body bias voltage) pairs that can operate the processor at a given frequency. Considering I_{BTBT} in leakage energy, we derive Eqs. (9) and (10) to find the energy optimal supply and body bias voltages that result in a given frequency and process technology. Equation (9) illustrates the relationship between the bias voltage and the derivative of the energy consumption per cycle. When the energy consumption derivative is zero, the energy consumption in that cycle is minimized. Associated with each frequency, there is a certain bias voltage that sets the derivate to zero in Eq. (9). Once the bias voltage and frequency are known, Eq. (10) can be used to derive the corresponding supply voltage.

$$\frac{\partial E_{\text{cyc}}}{\partial v_{\text{bs}}} = \begin{cases} L_g K_3 f^{-1} (K_1 V_{\text{bs}} + K_2)\, e^{K_3 V_{\text{bs}} + K_4} \\ \quad + L_g J_1 f^{-1} (J_2 V_{\text{bs}} - 1) e^{-J_2 V_{\text{bs}}} \\ \quad + 2C_{\text{eff}}(K_5 V_{\text{bs}} + K_6) \quad \text{if } V_{\text{dd}} > 0.5 \\ \frac{L_g}{2f}(K_3 K_5 e^{K_5 V_{\text{bs}} + 0.5 K_4} \\ \quad + 2(J_1 (J_2 V_{\text{bs}} - 1) e^{-J_2 V_{\text{bs}}})) \quad \text{otherwise} \end{cases} \quad (9)$$

$$V_{\text{dd}} = (L_d K_6 f - K_2 V_{\text{bs}} + V_{\text{th1}})/(1 + K_1) \quad (10)$$

Where E_{cyc} is the energy consumption per cycle and L_g is the number of logic gates in the circuit. The selection of the operating modes is an important issue for both performance and energy consumption. The processors with too few operating modes cannot completely exploit the execution slack. On the other hand, too many operating modes would introduce extra complexity for hardware designers with very little energy improvement.[12, 15]

We assume that our target processor can operate at 5 different clock frequencies, from 200 MHz up to 1 GHz at 200 MHz steps. We adopt the process technology and processor parameters from Predictive Technology Models[1]

and existing DVS-enabled commercial processors,[2] respectively. Using Eqs. (9) and (10), we obtain the energy optimal supply and body bias voltages corresponding to each frequency. Table I demonstrates the characteristics of the operating modes for our target processor in 90 nm.

We assume that the target processor uses clock gating to eliminate dynamic energy consumption, whenever it is waiting for memory on a cache miss. During the idle cycles, the processor is not clocked and thus, it does not dissipate any active energy. However, it does dissipate energy through leakage during the idle cycles.

4.2. Energy and Delay Overhead of Mode Switching

Each operating mode is composed of two distinct supply and substrate voltages. In order for the processor to switch between two operating modes, the voltage regulators have to charge or discharge large substrate and power rail capacitances. Ideally, increasing the voltage of a capacitance stores battery energy on the capacitor, and hence, it does not "dissipate" energy. Similarly, an ideally-efficient regulator can reduce a capacitor's voltage by taking charge off the capacitor, and storing it in an inductor or another capacitor.

Realistically however, DC–DC voltage conversion circuitry such as Buck converters dissipate energy to perform this function.[6] Thus, mode switching incurs both delay and energy penalty. Voltage regulator circuitry are used for both increasing and decreasing voltage. Therefore, the penalty exists for switching between any two modes. The magnitude of the delay and energy penalty would depend on the absolute difference between the corresponding voltages (supply or substrate) of the two modes, the corresponding capacitance, and the maximum current of the voltage regulator.

The energy overhead of mode switching is the sum of energy dissipated by supply and substrate voltage regulators, and depends on their efficiency. The energy dissipated by each voltage regulator is proportional to square of the voltage difference of the two modes times the charged capacitance.[6] Note that both increasing and decreasing voltage comes with an energy overhead in the regulator. The following equation formalizes this relation:

$$E_s = (1-\eta_{\mathrm{dd}})\cdot C_r \cdot |V_{\mathrm{dd}_1}^2 - V_{\mathrm{dd}_2}^2| + (1-\eta_{\mathrm{bs}})\cdot C_s \cdot |V_{\mathrm{bs}_1}^2 - V_{\mathrm{bs}_2}^2| \tag{11}$$

where E_s is the energy overhead of mode switching, C_r is the capacitance of the power rail and C_s is the capacitance of the substrate of the device. Efficiencies of the DC–DC converters for both supply and body voltage regulators are represented by η_{dd} and η_{bs}. V_{dd} and V_{bs} denote the supply and body substrate voltages, respectively. Note that no energy would be lost during DC–DC conversion by an ideal regulator.

Table I. Five processor operating modes at 90 nm.

Operating frequency (MHz)	1000	800	600	400	200
Supply voltage (V)	1.62	1.46	1.27	1.05	0.94
Bias voltage (V)	−0.069	−0.155	−0.236	−0.33	−0.44

The mode switching delay overhead associated with any of the two supply and substrate voltage scaling can be approximated as the time required to charge the corresponding capacitance using voltage regulators.[6] The overall delay overhead, however, is determined by the slower process of supply or substrate voltage adjustment, and can be written as:

$$L_s = \mathrm{Max}\left\{\frac{2\cdot C_r}{I_{r,\max}} \mid V_{\mathrm{dd}_1} - V_{\mathrm{dd}_2} \mid, \frac{2\cdot C_s}{I_{s,\max}} \mid V_{\mathrm{bs}_1} - V_{\mathrm{bs}_2} \mid\right\} \tag{12}$$

where $I_{r,\max}$ and $I_{s,\max}$ are the maximum possible current of the supply and substrate voltage regulators, respectively. The term $C_r \cdot \mid V_{\mathrm{dd}_1} - V_{\mathrm{dd}_2} \mid$ represents the electric charge difference of the supply regulator between two operating modes. Dividing $C_r \cdot \mid V_{\mathrm{dd}_1} - V_{\mathrm{dd}_2} \mid$ to $I_{r,\max}$ would give the switching delay caused by supply regulator, if the capacitor is charged with constant current of $I_{r,\max}$. The factor of 2 exists in the equation because of regulator's triangular current waveform.[6]

In our study, we model the energy and delay overhead of mode switching with the aforementioned equations assuming 90% efficiency for voltage regulators. For each technology, the capacitances are typical of embedded processors in that technology. For example, switching from mode 1 to mode 3 in 90 nm technology incurs 12.2 u joules energy overhead, and 80.5 u seconds delay overhead for our target processor.

5. LEAKAGE-AWARE COMPILATION

Compilers can minimize the application total energy consumption by statically assigning different execution traces to different processor operating modes (or simply modes) subject to meeting the deadline of the application. This can be viewed as inserting mode-switch instruction on selected control edges of the application before generating code. The challenge, however, is to determine the right set of control edges and operating modes during compilation. The energy and latency footprint of mode switching are substantially large. As a result, switching between modes have to be temporally spaced out to justify their overhead.

We formulate this problem using mixed-integer linear programming (MILP), considering energy and latency penalty of switching between modes, and assuming that each control flow edge will include a mode switching instruction. This is merely to expose the entire solution space to the MILP solvers, and not to suggest that every control edge will switch the processor mode. Realistically, the solver is forced to execute many temporally-close basic blocks in the same mode in order to avoid paying the high

penalty of frequent mode switchings. A similar approach has been used by Xie et al.[13] for DVS-enabled processor, however, their technique neglects the leakage contribution to total power consumption. We formulate the problem to consider both dynamic voltage scaling and adaptive body biasing.

Multiple control edges might lead to the same basic block. Assignment of mode switch instructions to control edges implies that different iterations of a basic block might be executed in different operating modes. However, subsequent iterations over a particular control flow edge would always switch the processor to a specific mode. Figure 4 shows a partial view of a program control flow structure, which illustrates sample branches, loops, and self-loops among the basic blocks.

The structure of the application and its typical execution traces can be extracted via statistical analysis or simulation-based profiling to determine information such as the average latency and the average energy consumption for each basic block under a particular mode, and the frequency of traversing edges. The extracted information are utilized to formulate the MILP instance, which can be solved by application of commercially available solvers. Once mode assignments for control flow edges of the application are decided, appropriate mode switching instructions can be inserted in the code, and executable binaries can be generated.

We associate a binary decision variable k_{bcm} to represent the operating mode on the edge (b, c) of the application CDFG. k_{bcm} is set to 1 if and only if the operating mode on the edge (b, c) is set to mode m. We use the constant E_{cm} to denote the average energy consumption of the basic

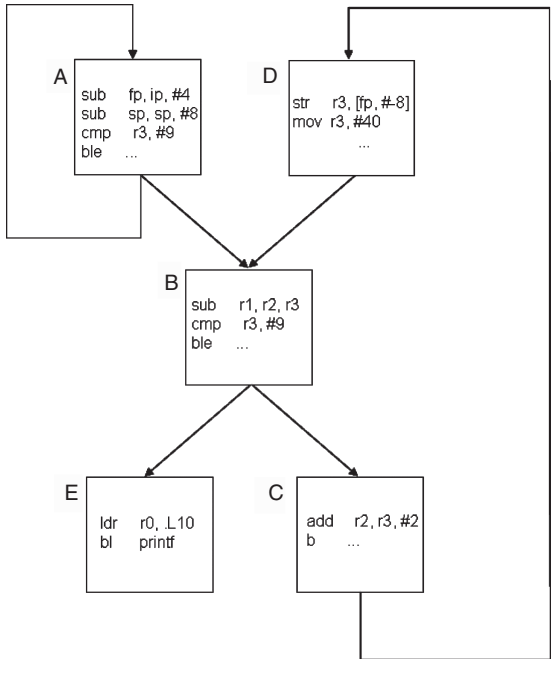

Fig. 4. Partial CDFG structure of an example program.

block c in mode m. Constant D_{abc} is used to represent the number of times basic block b is entered through edge (a, b) and exits through edge (b, c) (Fig. 4). D_{abc} represents the transition into and out of basic block b, which assists us in determining whether the two edges will incur a mode switch, or they should run in the same operating mode. The constant value of D_{abc} can be determined or estimated by application profiling. Similarly, G_{bc} denotes the number of executions of edge (b, c). Therefore, the total energy consumption of the application is given by:

$$\sum_{b=1}^{R}\sum_{c=1}^{R}\sum_{m=1}^{N} G_{bc}k_{bcm}E_{cm} + \sum_{a=1}^{R}\sum_{b=1}^{R}\sum_{c=1}^{R}\left(D_{abc}\times(1-\eta_{\mathrm{dd}})\times\sum_{m=1}^{N}(|k_{abm}V_{m,s}^2 - k_{bcm}V_{m,s}^2|C_r)\right) + \sum_{a=1}^{R}\sum_{b=1}^{R}\sum_{c=1}^{R}\left(D_{abc}\times(1-\eta_{\mathrm{bs}})\times\sum_{m=1}^{N}(|k_{abm}V_{m,b}^2 - k_{bcm}V_{m,b}^2|C_s)\right) \quad (13)$$

Where $V_{m,s}$ and $V_{m,b}$ are constants representing the supply and bias voltage under operating mode m. R is the number of the basic blocks in the control-flow graph, and N is the number of the operating modes of the microprocessor. The first term of Eq. (13) represents the energy consumption for execution of the basic blocks at their associated operating modes. The second and third terms are the switching energy penalties caused by DVS and ABB, respectively (Subsection 4.2). Hence, the objective of our optimization is to minimize Eq. (13) by manipulating the variables k_{bcm}, which would determine the proper mode for execution of each basic block.

For realtime applications, the compiled code has to guarantee an execution deadline. We model this constraint by inequality (14), which would prevent the MILP solver to generate solutions that violate the deadline constraint. The first term of (14) represents the latency for execution of the basic blocks at their associated operating modes. The second and third terms are the switching latency penalties caused by DVS and ABB, respectively (Subsection 4.2). The constraint forces the execution deadline of the application to be met:

$$\sum_{b=1}^{R}\sum_{c=1}^{R}\sum_{m=1}^{N} G_{bc}k_{bcm}T_{cm} + \sum_{a=1}^{R}\sum_{b=1}^{R}\sum_{c=1}^{R} D_{abc} \times \mathrm{Max}\Big(\sum_{m=1}^{N}(|\,k_{abm}V_{m,s} - k_{bcm}V_{m,s}\,|\,C_R) \sum_{m=1}^{N}(|\,k_{abm}V_{m,b} - k_{bcm}V_{m,b}\,|\,C_B)\Big) \le \mathrm{deadline} \quad (14)$$

where C_R and C_B are constants equal to $(2\cdot C_r)/I_{r,\max}$ and $(2\cdot C_s)/I_{s,\max}$, respectively. T_{cm} is the average latency of basic block c when run in mode m, and $V_{m,s}$ and $V_{m,b}$ are

constants representing the supply and body bias voltages in mode m.

Capturing G_{bc} and T_{cm} parameters is somewhat different for hard realtime as opposed to soft realtime applications. For hard realtime applications, which demand strict guarantees on meeting the deadline, the application has to be profiled with the input data that incur worst case execution time (WCET). Soft realtime applications, on the other hand, are only sensitive to average case execution latency in which case, the parameters should be captured using input data that incur average execution latency. The MILP formulation remains valid for both cases.

To guarantee that each basic block will be executed in exactly one mode, another MILP constraint is added. In this constraint, the summation of all k_{bcm} variables is forced to be equal to 1, which implies that exactly one of the operating modes is assigned to each control edge.

$$\sum_{m=1}^{N} k_{bcm} = 1 \tag{15}$$

The astute reader notices that the absolute and max function used in Eqs. (13) and (14) are not linear constraints. To linearize the max function in Eq. (14), we introduce a new variable that is greater than both terms inside the max function. In addition, we introduce variables e_{abc} to linearize the absolute function used in expressing the supply voltage energy overhead term (Eq. (13)):

$$-e_{abc} \leq \sum_{m=1}^{N} (k_{abm} V_{m,s}^2 - k_{bcm} V_{m,s}^2) \leq e_{abc} \tag{16}$$

Note that $V_{m,s}$ represents the supply voltage in mode m and is a constant. Three other variables and corresponding constraints are also added, to linearize the terms associated with supply voltage in (14), and with body bias voltage in (13) and (14).

For real life applications, the average basic block has about 5 to 10 instructions. Therefore, the delay and energy overhead of switching between two modes is substantially larger than the delay and energy of executing a basic block of average complexity. Therefore, it is not practical to execute mode switch instructions for each edge of the application CDFG. It is beneficial to point out that although our formulation starts with assigning a distinct mode switch instruction to each edge, the MILP solver ends up assigning many consecutive CDFG edges to the same mode. Thus, many edges will have redundant mode switch instructions that can be removed for further optimization. This issue will be discussed at length in the experiments section.

6. EMPIRICAL VALIDATION

6.1. Setup and Methodology

In order to verify the effectiveness of our intraprogram combined voltage scaling and body biasing technique, we have developed a compilation flow to generate executable code for our target processor. Figure 5 illustrates our experimental setup. We have instrumented widely used compiler infrastructure, cycle-accurate performance and energy simulators to obtain the required profiling information for each benchmark application. Subsequently, we generate the MILP problem and invoke a commercial solver to obtain the optimized operating mode for each control flow edge of the CDFG. The MILP solution is fed back to our compiler, which inserts corresponding mode switch instructions on control flow edges before code generation. Finally, the generated code is simulated using cycle-accurate simulators to measure its energy consumption, and to ensure that it meets the deadline constraint.

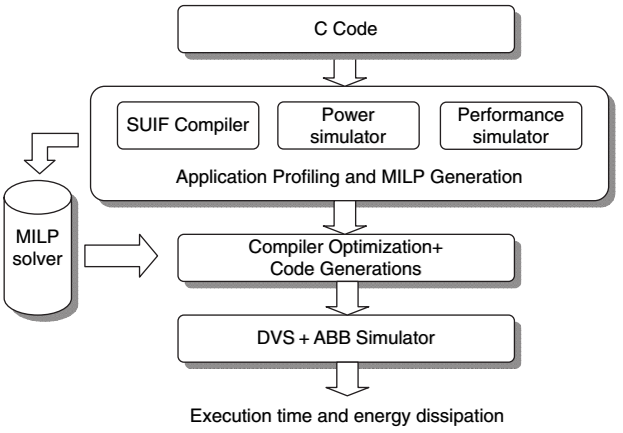

Fig. 5. The setup of experiments for leakage-aware compilation.

The first stage of our compilation methodology is to profile a given application to obtain the required information for our proposed MILP formulation. The required information include average delay and energy dissipation of program basic blocks in each operating mode (E_{cm} and T_{cm} in Eqs. (13) and (14)), and execution frequency of control flow edges with respect to preceding active basic block (G_{bc} and D_{abc} in Eq. (13)). The front end of our compiler is based on the MachineSUIF compiler framework.[23] We utilized MachineSUIF to generate CDFG representation of a program, and extract part of the profiling information that relate to execution frequency of basic blocks.

Furthermore, our method assumes knowledge of the average latency and energy dissipation of basic blocks under each operating frequency. Note that the memory is not synchronous with processor, and has a rather constant access time. Therefore, frequency scaling leads to a different set of cache misses/hits, and hence, different number of cycles to execute the program. We instrumented the simplescalar simulator[34] to report the average latency of each basic block. Also, we modified XTREM simulator[14] to report average overall energy dissipation of basic blocks.

Capturing program characteristics such as average latency or energy dissipation of basic blocks, execution frequency of basic blocks and control flow edges directly

depend on the input data used for profiling. We resort to the input data associated with the worst case execution time (WCET), because we must guarantee that the generated code would meet the execution deadline. If the hard realtime constraint did not exist, we could have used a more relaxed input data to explore a larger solution space. In general, one needs to study sensitivity of MILP instance to input data, in order to quantitatively analyze the choice of input data on the quality of generated code. While absent in the present work, we hope to tackle this issue as part of our future research agenda.

We defined the mode-switch instruction in the instruction decoding tree of our XTREM-based simulator. We also defined the mode switch instruction in our cross-compiler/assembler. Therefore, we can both generate and simulate mode switching instructions in the assembly code. As simulator simulates the mode switch instruction, it changes the supply voltage, bias voltage, and frequency parameters of the simulator and calculates the corresponding energy consumption and execution time. Once the necessary profiling information are gathered, we generate the MILP instance. The CPLEX package is used to solve the problem instances.

We have selected seven applications of different complexity from MiBench[22] embedded application suite as our testbenches. The selected applications represent several application domains of embedded systems, including networking, automotive, telecom and security. Table II reports the characteristics of the selected applications. The selected application domains justify the need for deadline constraint and realtime operation of the generated code.

In order to investigate the effect of deadline relaxation on the quality of different frequency scaling methods, we have carried out our experiments using five different deadlines for each application. Figure 6 visualizes the relative location of deadlines in comparison to baseline execution times. For each benchmark application, we executed the program at all different frequency modes of the processor (with no DVS or ABB mechanism) to obtain the baseline execution time of the application at each frequency. The first four deadlines are determined by averaging the adjacent execution times. For example, the first deadline is equal to the average execution time at 1 GHz and 800 MHz frequencies, with no frequency scaling mechanism. The last (fifth) deadline is set to 95% of the execution time at the slowest mode, i.e., running the processor at 200 MHz.

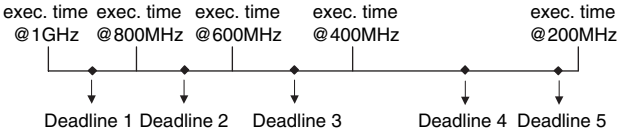

Fig. 6. Relation of deadlines to baseline execution times.

After solving the MILP problem instance, the appropriate execution mode for running each control flow edge is known. The second stage of the experiment is to insert mode switch instructions to control flow edges of the application. Our compiler inserts new basic blocks containing the appropriate mode switch instructions to control flow edges of the CDFGs. Since mode switching delay and energy penalty is significantly larger than that of the typical basic block, it is likely that MILP solver assigns two consecutive basic blocks to the same mode. In that case, the two consecutive mode switch instructions select the same operating mode, and hence, the second instruction is redundant.

Figure 7 shows two examples of redundant operating mode assignment. In the left example, basic block *C* will always be executed under the operating mode of basic block *B*. Therefore we can safely remove the mode switch basic instruction on the edge between basic block *B* and *C*. In the right example, the mode switch basic block on the edge from *E* to *D* is redundant. We found that the negative impact of redundant mode switch instructions due to introduction of extra delay and energy penalty, could be substantial. Furthermore, insertion of new basic blocks changes the CFG structure of the program. Careless insertion of basic block introduces redundant jump/branch instructions and further deteriorates the quality of the generated code. Redundant mode switches can be optimized using standard compiler optimization techniques such as dead-code elimination. We apply copy propagation, constant propagation, and dead code elimination to remove redundant mode switches, and improve the performance and energy dissipation of the generated code.

We have modified XTREM,[14] a power simulator for Intel XScale DVS-enabled core, to estimate the energy dissipation of the generated code on the processor with

Table II. Testbench applications and their characteristics.

Benchmark	Application domain	# Basic block	Exec. time @1 GHz	Exec. time @800 MHz	Exec. time @600 MHz	Exec. time @400 MHz	Exec. time @200 MHz	Average MILP solution time
adpcm	Telecom	33	12.45	17.27	22.31	33.46	62.77	5.12
sha	Security	32	11.57	14.08	20.42	28.72	58.89	5.56
dijkstra	Network	36	17.4	21.17	32.54	42.3	84.63	5.25
patricia	Network	138	28.43	38.01	52.14	74.61	146.21	184
susan	Automotive	203	23.45	31.36	43.14	62.47	121.45	1586
jpeg-dec	Consumer	212	25.90	64.25	45.41	64.33	129.39	1614
gsm-dec	Telecom	556	31.23	38.86	53.51	78.45	157.73	22455

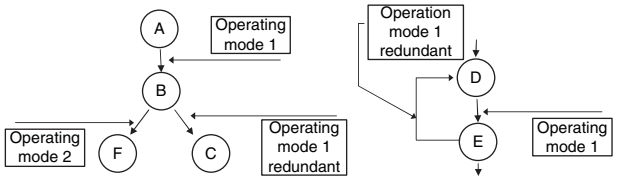

Fig. 7. Examples of the redundant operating mode assignments.

DVS+ABB capabilities. Our simulator recognizes mode switch instructions and tracks the execution time, leakage power and active power under frequency scaling. For each frequency mode, a particular pair of supply and body bias voltages is used. For a given frequency, the DVS-enabled processor has zero body bias, and hence, its supply voltage is different from (higher than) the DVS+ABB enabled processor. The optimal supply and bias voltage for each frequency are obtained using Eqs. (9) and (10). The developed simulator also considers the energy and delay penalty of switching between modes. Our simulator takes into account the impact of cache memory, and measures the latency and energy impact of cache misses. We do not consider the energy dissipation of the off-chip memory.

6.2. Energy Savings

We implemented the experimental flow depicted in Figure 5 and generated code for seven applications listed in Table II. To examine the impact of execution deadline and technology scaling, each application testbench is experimented under twenty different settings, i.e., five different execution deadlines (Fig. 6) and four different process technologies. The models and parameters of the process technology are adopted from Berkeley predictive technology models.[1] The reported execution time and energy dissipation are the numbers estimated from the generated code, using cycle-accurate simulation of the processor under mode parameters.

Table II presents the baseline execution time of the applications. These execution times are simulated for the processor running at a particular operating frequency without the frequency scaling mechanism. For example, execution time at 800 MHz would be the most energy efficient setting of the processor to meet deadline 2, with no frequency scaling. The table also reports the average (over twenty different settings) MILP solution time for each application in seconds.

Comprehensive experiments are carried out which, generate data for all application benchmarks under four technology nodes (130 nm, 90 nm, 65 nm and 45 nm) and five possible deadlines. For each setting, the energy consumption of the baseline execution, dynamic voltage scaling, and combined DVS and ABB are recorded.[a]

[a]The amount of generated data is too large to entirely report in this manuscript. We would like to seek reviewers' opinion on most important views of the data that could be rendered within the page limitation.

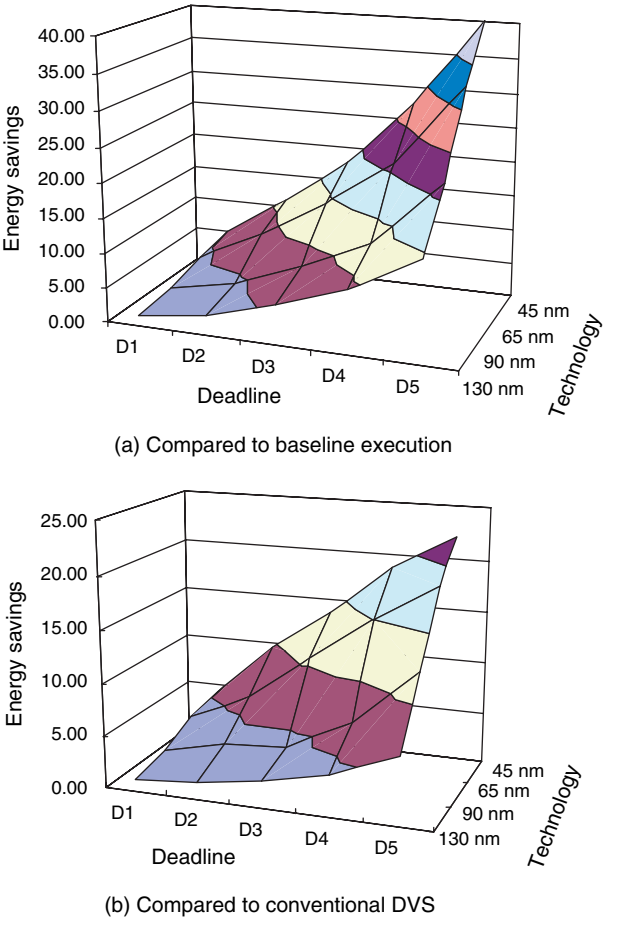

Fig. 8. Average energy savings of DVS+ABB.

Figure 8(a) summarizes the data using a three dimensional space of average energy savings, deadline, and process technology. The chart illustrates the DVS+ABB improvement in energy dissipation over baseline execution of the code. In baseline execution, processor is constantly run at the minimum frequency that meets the deadline. All improvement percentages are calculated with respect to corresponding baseline case to provide a fair ground for comparison of different experiments. Note that our notion of baseline is different from most of previous work that run the processor at the fastest frequency (1 GHz in our case) in baseline mode. As a result, the improvement numbers might seem relatively small. Similar to Figures 8(a, b) compares the average energy dissipation of DVS+ABB with conventional DVS, over various deadline and process technologies.

As Figures 8(a and b) suggest, the energy savings significantly increase with the relaxation of deadline, for a given process technology. Deadline relaxation increases the available timing slack and solution space and hence, it becomes easier for our compilation method to switch between modes while meeting the deadline. For example in 65 nm process, gradual relaxation of the timing constraint from deadline 1 to deadline 5 leads to 3.9%, 8.41%,

14.31%, 20.95% and 32.65% improvement in energy dissipation over baseline execution.

DVS+ABB is aware of the leakage's contribution in total dissipated energy for each technology node. It selects the operating modes such that the overall energy dissipation, including leakage, is optimized. Consequently, DVS+ABB consistently outperforms conventional DVS. It is interesting to note that DVS+ABB performs only slightly better than DVS for 130 nm, where, leakage energy is negligible. However with the shrinkage of the device sizes, the leakage energy increases exponentially. As a result, our method can exploit more energy saving in advanced technologies. For example considering deadline 4, our approach achieves 4.06%, 6.88%, 15.01% and 18.68% energy improvement over conventional DVS, for 130 nm, 90 nm, 65 m and 45 m, respectively.

6.3. Scaling Limitation of ABB

It is interesting to note the *rate of change* in improvement of DVS+ABB over DVS with scaling. This is depicted visually as the slope of the plane with respect to technology scaling axis in Figure 8. Scaling from 130 nm to 90 nm and from 90 nm to 65 nm, the rate of improvement consistently increases. For example under deadline 4, the improvement of DVS+ABB over DVS going from 130 nm to 90 nm is increased by about 2.8%. This improvement grows to over 7% when we scale from 90 nm to 65 nm. However, the rate is slowed down again after 65 nm. In the example of deadline 4, the improvement is about 3.6% when we scale from 65 nm to 45 nm.

This trend reaffirms our earlier discussion in Section 3 on scaling limitations of ABB. Scaling up to 65 nm, the share of leakage in total energy increases but BTBT current does not become excessively large. The BTBT leakage current exponentially increases with technology scaling and thus, effectiveness of reverse body biasing is curtailed with scaling beyond 65 nm. This is depicted by small change in improvement of DVS+ABB over DVS when we scale from 65 nm to 45 nm. It is expected that the plane in Figure 3 would plateau, and hence DVS+ABB would be the same as DVS, with further scaling.

Note that our DVS+ABB scheme cannot become worse than DVS. For a given frequency, we calculate body bias voltage to minimize overall energy consumption in that frequency. With scaling beyond 65 nm, the share of BTBT current in total leakage current grows. Furthermore, BTBT exponentially increases with reverse body voltage. As a result, our analytical model would reduce the reverse body bias with further scaling to address the issue. We already observe the descending rate of body bias with scaling in existing operating modes. In the extreme case, our method will eliminate reverse body bias and will become identical to conventional DVS schemes.

6.4. Mode Switching Overhead

As discussed in Section 4.2, the energy and latency overhead of mode switching is substantially larger than that of a typical (5∼10 instructions) basic block. This might raise concerns about our initial formulation that adds a mode switch instruction to every edge of the application CDFG. We would like to emphasize that our ILP formulation considers the energy and latency overhead for switching between two different modes as a result of which, most of the consecutive mode switch instructions operate the processor in the same mode. Hence, they do not incur any switching energy or latency penalty. The dead-code elimination pass applied before code generation eliminates the redundant modes and hence, the generated code only has a few mode switch instructions on selected edges of the CDFGS.

Table III reports the statistics on mode switches in application benchmarks under deadline 4 and in 65 nm technology. The first, second and third columns of the table list application benchmarks, and their complexity in terms of the number of basic blocks and control flow edges in their CDFG. The forth column reports the number of mode switch instructions inserted in the generated code. The column "annotation ratio" is the percentage of the control flow edges that are assigned a mode switching instruction in the generated code. Note that annotation ratio is on the order of a few percent (2%–9.1%), which implies that only a small number of selected edges will perform mode switching in the generated code.

More importantly, the selected edges are infrequently executed. Thus, the dynamic energy and latency overhead associated with mode switching is small. The sixth and seventh column of Table III report the dynamic count of mode switch instructions and application instructions, respectively. Dynamic count of mode switchings is on the order of tens to at most two hundred for our test cases, while several billions of regular instructions are executed in the applications. Remember that a mode switching incurs a large latency (on the order of ten thousands cycles) and hence, dynamic count of mode switches should be much smaller than dynamic instruction count.

Finally, the last two columns report the percentage of the overall energy and latency that is spent in mode switching. In other words, the columns show the energy and latency contribution of mode switching overhead in total application energy dissipation or execution latency. The energy contribution of mode switching overhead is only about 0.001% on average. The latency contribution is about 0.73%, which is quite small. In summary, Table III shows that the generated code is far from invoking a mode switch on every control flow edge of the application. However, we had to start with such an assumption to formulated the problem using MILP.

The main memory is a major contributor to total energy dissipation of many embedded systems. Although we only

Table III. Mode switching statistics under deadline 4 in 65 nm.

Benchmark	# of basic blocks	# of control flow edges	# of mode switch instructions	Annotation ratio (%)	Dynamic # of mode switches	Dynamic # of instructions (Billions)	Energy penalty (%)	Latency penalty (%)
adpcm (coder)	33	44	4	9.1	51	2.58	4.81×10^{-4}	0.34
patricia	138	186	11	5.9	132	5.93	0.98×10^{-3}	0.87
dijstra	36	48	4	8.3	59	3.84	3.91×10^{-4}	0.25
susan (corner)	203	328	13	4.0	162	4.79	3.12×10^{-3}	1.09
sha	32	57	4	7.0	55	2.49	4.14×10^{-4}	0.33
gsm-dec	556	833	17	2.0	221	6.45	2.74×10^{-3}	1.26
jpeg-dec	212	301	14	4.7	181	5.21	1.55×10^{-3}	0.99

focus on energy dissipation of the embedded processor, we argue that our compiler will have minimal impact on energy dissipation of the main memory. According to the data in Table III, our compiler will have negligible affect on both code size and executed instructions. Table III implies that no mode switching instruction is left in critical loops or other frequently-executed blocks that typically account for 80–90% of program execution runtime. As a result, static and dynamic behavior of the code is very similar to that of code without mode switching instructions.

6.5. Impact of Power Rail and Substrate Capacitance

The mode switching energy and latency overhead is due to the capacitance of both power rail and substrate of the microprocessor. For switching to a high-frequency operating mode, processor voltage regulation circuitry needs to charge the capacitances, which takes both time and energy. Mode switching latency and energy overhead are the main factors in limiting the energy savings of frequency scaling methods.

It is important to measure the impact of switching overhead on compiler's energy savings. This translates to investigating the effect of power rail and substrate capacitance on energy savings. The study can assist in evaluation of the usability of our technique for different kinds of microprocessor. Since it is not practical to accurately estimate the capacitances for future processors, we use the typical values for current embedded processors along with two scaling factors of 10 and 10^{-1}.

The typical power rail capacitance and substrate capacitance values for today's embedded processors are 20 uF and 80 uF, respectively. We experiment the impact of the mode switching overhead by using 2 uF, 20 uF, and 200 uF for power rail capacitance, and 8 uF, 80 uF, and 800 uF for substrate capacitance. The simulation results for benchmarks *susan* and *patricia* under deadline 4 are illustrated in Figures 9(a–c).

The figures demonstrate the average energy savings of our DVS+ABB compilation technique versus baseline execution or DVS-only compilation. Figure 9(b) shows the situation in which, we have low penalty (2 uf) for supply voltage scaling. Figure 9(c) is the case for today's typical embedded processors in which, the power rail capacitance has its typical value (20 uf). Figure 9(a) depicts the savings for a processor with large power rail capacitance (200 uf).

Enlarging power rail or substrate capacitance increases the mode switch energy penalty, and hence, mode switch instructions become *more expensive* for the compiler to use. Consequently, the generated code will contain less mode switches and part of the energy savings compared to baseline execution is diminished. In the extreme case, where capacitances are very large and mode switching is

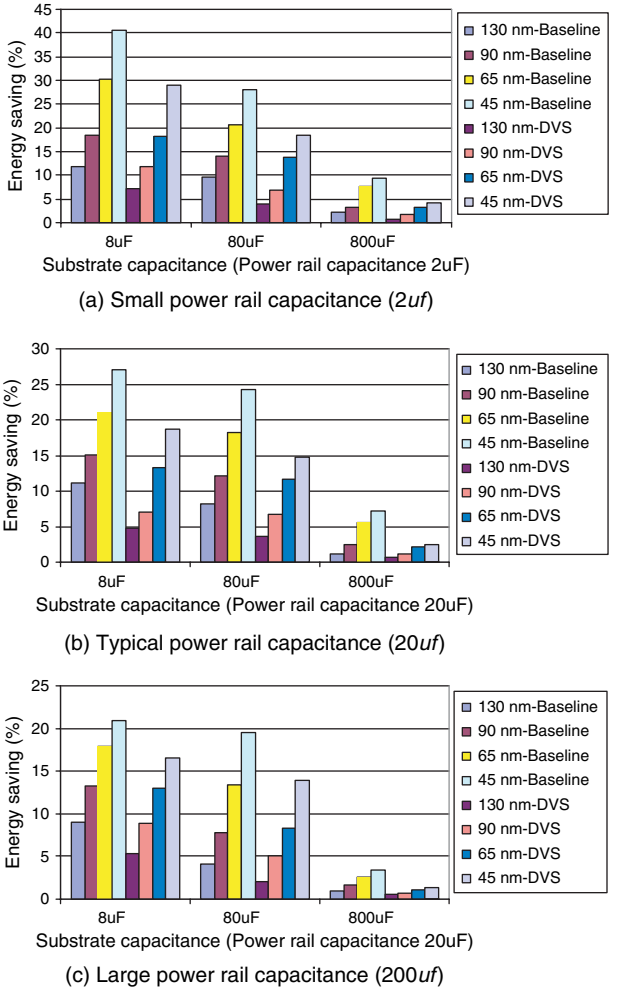

Fig. 9. Variations in energy savings with respect to substrate capacitance.

substantially expensive, the compiler cannot use any mode switch instruction and thus, the generated code will dissipate the same amount of energy as baseline execution. On the other hand, smaller capacitances decrease the mode switch penalty, which allows the compiler to perform finer grain assignment of regions of code to operating modes. Hence, the energy savings are improved.

7. ACCELERATING MILP SOLVING TIME

MILP is a well-known NP-hard problem and thus, it is unlikely that an algorithm with polynomial time complexity exists to optimally solve arbitrary instances of MILP. MILP solvers often employ branch and bound heuristics to explore the solution space more efficiently and accelerate their runtimes. Nevertheless, the running time of the solver for some problem instances can be prohibitive. As shown in Table II, the solving time for problem instances of large applications such as *susan* and *jpeg-dec*, is not reasonable for integration into a real life compiler. In this section, we develop heuristics for reducing the MILP solving time at the price of reasonable loss in energy savings.

The runtime of MILP solver exponentially increases with growth of the problem instance complexity. The complexity of a MILP problem instance depends on the number of both variables and constraints, as well as the problem structure. An intuitive approach to accelerate the solving time is to eliminate some of the variables and constraints that have negligible or small effect on the quality of the overall solution. Elimination of selected MILP variables will bring an unavoidable energy downgrade, because the reduced MILP instance with less variables cannot necessarily deliver the solution to the original MILP instance. However, judicious elimination of variables and corresponding constraints will minimize the loss in energy savings.

Intuitively, basic blocks with small contributions to the application energy (compared to the average contribution of basic blocks) do not substantially deteriorate the compiler's energy savings if executed in a different mode. Hence, a good set of candidate variables for elimination are the mode select variables (k_{abm} in Section 5) for such basic blocks. This group of basic blocks includes those that are very short and infrequently executed.

Our heuristic algorithm for accelerating the MILP solution time is based on this idea. We find the basic blocks with small share in energy dissipation of the application. The energy contribution of each basic block is estimated using our simulation and profiling framework. If the contribution of the basic block to overall energy consumption is small, it will be executed in the same operating mode as its more frequently executed preceding basic block. That is, its corresponding k_{bcm} variables are set to be equal to another variable such as k_{abm}. This process eliminates some of the independent variables in the problem instance, and reduces its complexity.

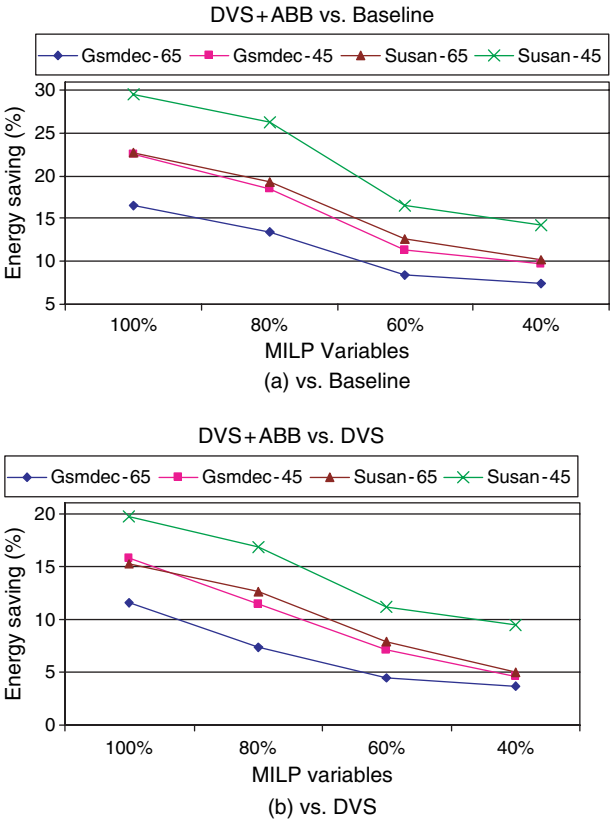

Fig. 10. Impact of MILP instance complexity reduction on energy savings.

Figures 10(a and b) illustrate the impact of our MILP instance complexity reduction algorithm on the energy savings of our compilation technique. The figure depicts the energy savings of our ABB+DVS compilation versus baseline execution and DVS-only compilation for benchmarks *susan* and *gsm-dec* under deadline 4, and in 65 nm and 45 nm technologies. The numbers on the X axis represent the percentage of the independent variables that are maintained in the problem formulation. For example, the 60% point on X axis refers to elimination of 40% of independent variables that correspond to 40% of basic blocks with least contribution to application energy dissipation.

Figure 11 illustrates the effect of variable elimination on MILP solver runtime. The X axis in the figure uses the same notion as Figure 10(a). The solver running time is demonstrated on the Y axis in Figure 11. For each case, the running time of the solver is normalized to its running time when no variable is eliminated, i.e., 100% of the variables are present in the formulation.

Collective analysis of Figures 10(a, b) and 11 delivers a practical trade-off between energy savings and optimization runtime. For example in case of *susan*, the figures suggest that by filtering out 40% of least important independent variables (60% point on X axis) the runtime can be improved by about an order of magnitude, while

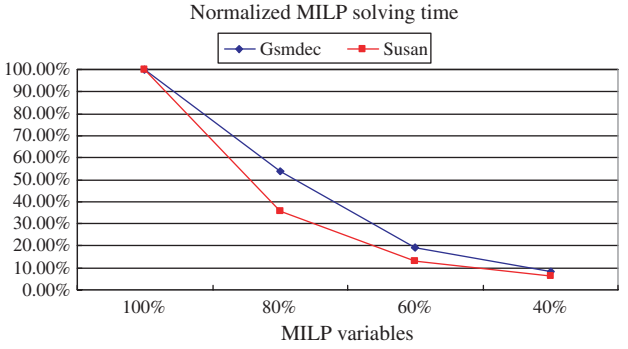

Fig. 11. Impact of MILP instance complexity reduction on MILP solving time.

degrading the energy savings by about 10%. This translates to solver runtime of under 3 minutes, which is quite affordable for compiling a realtime embedded application. If faster runtimes are desired, larger sacrifice in energy savings can be made to accelerate the solver even further.

8. CONCLUSIONS

We present a methodology to combine dynamic voltage scaling and adaptive body biasing during compilation of an application targeting a DVS+ABB enabled embedded processor. Compiler-level analysis is particularly useful for embedded systems that demand light-weight operating systems. Moreover, compilers can exploit program execution trace information that are not visible to the OS. We develop a compiler framework that generates code for a DVS+ABB enabled processor. Experimental results advocating the effectiveness of our approach, show that our compilation techniques reduce the overall energy consumption significantly. We discuss that reverse body biasing has practical limitations with scaling beyond 45 nm in which case, our DVS+ABB enabled processor will look like a conventional DVS enabled processor. We also develop some heuristics to reduce the MILP solving time of our proposed methodology. By using our heuristic, our method can be applied to most of the embedded applications. Future work includes sensitivity analysis of our formulation with respect to input data and corresponding profiling information.

References

1. http://www-device.eecs.berkeley.edu/ptm/introduction.html.
2. http://www.intel.com/design/intelxscale.
3. http://www.transmeta.com/longrun2/index.html.
4. A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, and A. Veidenbaum, Profile-based dynamic voltage scheduling using program checkpoints. *Design Automation and Test in Europe* 168 **(2002)**.
5. A. Kesharvarzi, S. Narenda, S. Borkar, C. Hawkins, K. Roy, and V. De, Technology scaling behavior of optimum reverse body bias for leakage power reduction in ICs. *International Symposium Low Power Electronics and Design*, August **(1999)**, pp. 252–254.
6. T. D. Burd and R. W. Brodersen, Design issues for dynamic voltage scaling. *International Symposium on Low Power Electronics and Design* **(2000)**, pp. 9–14.
7. C.-H. Hsu and U. Kremer, The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. *Conference on Programming Language Design and Implementation*, June **(2003)**, pp. 38–48.
8. L. T. Clark, E. J. Hoffman, J. Miller, M. Biyani, Y. Liao, S. Strazdus, M. Morrow, K. E. Velarde, and M. A. Yarch, An embedded 32-b microprocessor core for low-power and high-performance applications. *IEEE Journal of Solid-State Circuits* 36, 1599 **(2001)**.
9. L. T. Clark, M. Morrow, and W. Brown, Reverse-body bias and supply collapse for low effective standby power. *IEEE Transactions on Very Large Scale Integration Systems* 12, 947 **(2004)**.
10. D. Duarte, N. Vijaykrishnan, M. J. Irwin, H.-S. Kim, and G. McFarland, Impact of scaling on the effectiveness of dynamic power reduction schemes. *Proceeding of International Conference on Computer Design*, September **(2002)**, pp. 382–387.
11. D. Duarte, Y. Tsai, N. Vijaykrishnan, and M. J. Irwin, Evaluating run-time techniques for leakage power reduction. *Proceeding of International Conference on VLSI Design*, January **(2002)**, pp. 31–38.
12. D. Lackey, P. Zuchowski, T. Bednar, D. Stout, S. Gould, and J. Cohn, Managing power and performance for system-on-chip designs using voltage islands. *IEEE/ACM International Conference on Computer Aided Design* **(2002)**, pp. 195–202.
13. F. Xie, M. Martonosi, and S. Malik, Intraprogram dynamic voltage scaling: Bounding opportunities with analytic modeling. *ACM Transactions on Architecture and Code Optimization* 1, 1 **(2004)**.
14. G. Contreras, M. Martonosi, J. Peng, R. Ju, and G. Y. Lueh, XTREM: A power simulator for the Intel XScale core. *ACM SIGPLAN Notices* 39, 115 **(2004)**.
15. M. Hamada, Y. Ootaguro, and T. Kuroda, Utilizing surplus timing for power reduction. *IEEE Conference on Custom Integrated Circuits* **(2001)**, pp. 89–92.
16. P.-K. Huang and S. Ghiasi, Leakage-aware intraprogram voltage scaling for embedded processors. *Design Automation Conference* **(2006)**, pp. 364–369.
17. P.-K. Huang and S. Ghiasi, Efficient and scalable compiler-directed energy optimization for realtime applications. *Design Automation and Test in Europe* 785 **(2007)**.
18. P.-K. Huang and S. Ghiasi, Efficient and scalable compiler-directed energy optimization for realtime applications. *ACM Transactions Design Automation of Electronic Systems* 12 **(2007)**.
19. J. T. Kao, M. Miyazaki, and A. P. Chandrakasan, A 175-mv multiply-accumulate unit using an adaptive supply voltage and body bias architecture. *Journal of Solid-State Circuits* 37, 1545 **(2002)**.
20. K. A. Bowman, B. L. Austin, J. C. Eble, X. Tang, and J. D. Meindl, A physical alpha-power law MOSFET model. *IEEE Journal of Solid-State Circuits* 34, 1410 **(1999)**.
21. A. Keshavarzi, C. Hawkins, K. Roy, and V. De. Effectiveness of reverse body bias for low power cmos circuits. *NASA Symposium on VLSI Design* **(1999)**, pp. 2.3.1–2.3.9.
22. M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, and T. Mudge, Mibench: A free, commercially representative embedded benchmark suite. *Proceeding of the IEEE 4th Annual Workshop on Workload Characterization*, December **(2001)**, pp. 3–14.
23. M. D. Smith and G. Holloway, An introduction to machine SUIF and its portable libraries for analysis and optimization. Technical report, Division of Engineering and Applied Sciences, Harvard University **(2002)**.
24. M. J. Chen, H. T. Huang, C. S. Hou, and K. N. Yang, Back-gate bias enhanced band-to-band tunneling leakage in scaled MOSFETS. *IEEE Electron Device Letters* 19, 134 **(1998)**.
25. M. R. Stan, Optimal voltages and sizing for low power. *Intl. VLSI Design Conf.* **(1999)**, p. 428.

26. N. AbouGhazaleh, D. Mossé, B. R. Childers, R. G. Melhem, and M. Craven, Collaborative operating system and compiler power management for real-time applications. *IEEE Real Time Technology and Applications Symposium* **(2003)**, pp. 133–143.
27. S. Narendra and A. Chandrakasan, Leakage in Nanometer CMOS Technologies, Springer **(2006)**.
28. C. Neau and K. Roy, Optimal body bias selection for leakage improvement and process compensation over different technology generations. *International Symposium on Low Power Electronics and Design* **(2003)**, pp. 116–121.
29. P. Ko, J. Huang, Z. Liu, and C. Hu, BSIM3 for analog and digital circuit simulation. *IEEE Symposium on VLSI Technology CAD* **(1993)**, pp. 400–429.
30. P. Pillai and K. G. Shin, Real-time dynamic voltage scaling for low-power embedded operating systems. *ACM Symposium on Operating System Principles* **(2001)**, pp. 89–102.
31. R. Gonzales, B. M. Gordon, and M. A. Horowitz, Supply and threshold voltage scaling for low power CMOS. *Journal of Solid-State Circuits* 32, 1210 **(1997)**.
32. S. F. Huang, C. Wann, Y. S. Huang, C. Y. Lin, T. Schafbauer, S. M. Cheng, Y. C. Cheng, D. Vietzke, M. Eller, C. Lin, Q. Ye, N. Rovedo, S. Biesemans, P. Nguyen, R. Dennard, and B. Chen, Scalability and biasing strategy for CMOS with active well bias. *Symposium on VLSI Technology. Digest of Technical Papers* **(2001)**, pp. 107–108.
33. S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design* **(2002)**, pp. 721–725.
34. T. Austin, E. Larson, and D. Ernst, SimpleScalar: An infrastructure for computer system modeling. *Computer* 35, 59 **(2002)**.
35. T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, A dynamic voltage scaled microprocessor system. *IEEE Journal of Solid-State Circuits* 35, 1571 **(2000)**.
36. W. Kim, J. Kim, and S. L. Min, Dynamic voltage scaling algorithm for fixed-priority realtime systems using work-demand analysis. *International Symposium on Low Power Electronics and Design* **(2003)**, pp. 396–401.
37. L. Yan, J. Luo, and N. K. Jha, Joint dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24, 1030 **(2005)**.

Po-Kuan Huang

Po-Kuan Huang *received the B.S. Degree in Electrical Engineering from National Cheng Kung University, Tainan, Taiwan, R.O.C., in 2002. He is currently working toward the Ph.D. degree at the University of California, Davis. His research interests are in the areas of embedded system design and design automation for electronic system.*

Soheil Ghiasi

Soheil Ghiasi *received his B.S. from Sharif University of Technology, Tehran, Iran in 1998, and his M.S. and Ph.D. in Computer Science from University of California, Los Angeles in 2002 and 2004, respectively. He received the Harry M. Showman prize from UCLA College of Engineering in 2004. Currently, he is an assistant professor in the department of electrical and computer engineering at the University of California, Davis. His research interests include different aspects of Embedded and Reconfigurable system design.*