Leakage-Aware Intraprogram Voltage Scaling for Embedded Processors

Po-Kuan Huang
Department of Electrical and Computer Engineering
University of California, Davis
Davis, CA
pohuang@ece.ucdavis.edu

Soheil Ghiasi
Department of Electrical and Computer Engineering
University of California, Davis
Davis, CA
soheil@ece.ucdavis.edu

ABSTRACT

With scaling of technology feature sizes, the share of leakage in total power consumption of digital systems continues to grow. Conventional dynamic voltage scaling (DVS) techniques fail to accurately address the impact of scaling on system power consumption and hence, are incapable of achieving energy efficient solutions. To overcome this problem, we utilize adaptive body biasing (ABB) to adjust transistors' threshold voltage at runtime. We develop a leakage-aware compilation methodology that targets embedded processors with both DVS and ABB capabilities. Our technique has the unique advantage of jointly optimizing active and leakage energy dissipation. Considering the delay and energy penalty of switching between operating modes of the processor and under deadline constraint, our compiler improves the energy consumption of the generated code by average of 13.07% and up to 30.26% at 90nm. While our technique's improvement in energy dissipation over conventional DVS is marginal (4.54%) at 130nm, the average improvement continues to grow to 7.8%, 15.94% and 29.56% for 90nm, 65nm and 45 technology nodes, respectively.

Categories and Subject Descriptors: B.8.2 [Hardware]:Performance Analysis and Design Aids

General Terms: Algorithms,Management,Performance

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 contributer to total power dissipation of a CMOS design. Quadratic dependence of active power on supply voltage, along with the lower oder impact of supply voltage on clock frequency motivated the idea of frequency and supply voltage scaling for processors [20]. In this scheme the operating frequency and supply voltage 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 [5]. Figure 1 illustrates the trend of active and leakage power consumption 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 total power consumption for a given frequency [18]. Unlike traditional voltage scaling (DVS), combined dynamic voltage scaling and adaptive body biasing (DVS+ABB) has the potential of jointly optimizing active and leakage power. Exploiting the energy savings potentials of this technique, greatly depends on the operating system and compiler algorithms that control the operating modes of the processor.

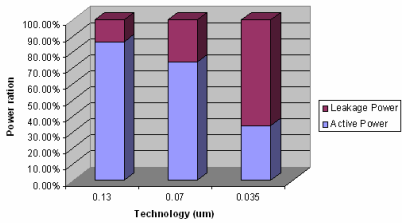

Figure 1: Estimation of active and leakage power variation across the technologies [5].

In this paper, we present a compilation methodology that targets embedded processors with joint DVS and ABB capabilities. By statically inserting mode switch instructions between some of the basic blocks of the code, our compiler generates code that is optimized for overall energy consumption. Moreover, the generated code is guaranteed to meet the deadline constraint for specified input patterns. Compared to baseline compilation, our compiler improves the energy consumption of the processor by 8.8%, 13.1%, 23.4% and 38.2% for 130nm, 90nm, 65nm and 45nm technologies, respectively. Compared to traditional DVS-only optimization, we improve the average energy consumption by 4.5%, 7.8%, 15.9% and 29.6% for the four aforementioned technologies.

2. RELATED WORK

Combined DVS and ABB optimization has been considered previously. In [18], a combined DVS+ABB technique is proposed for unrelated tasks. Also, an expression for obtaining the optimal tradeoff between bias voltage and supply voltage is derived. Yan

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

et al. proposed an algorithm for a task graph with real-time constraints [9]. None of the previous work on combined DVS+ABB have considered intra-task level optimization, which requires program structure analysis.

Several research efforts have proposed static intraprogram voltage scaling [6, 15, 3, 4]. An analytical study of potential power savings using intraprogram DVS is reported in [6]. The authors also propose an ILP-based based approach whose savings come reasonably close to the analytical bounds. In [3], 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[4]. Compiler and operating system level optimization are coordinated in [15]. None of these techniques consider leakage power, and the effect of technology scaling on the validity of their results. We utilize the generic power model derived in [18], and perform intraprogram simultaneous DVS and ABB. Using cycle-accurate simulators, we estimate the energy consumption of some embedded application to demonstrate the efficiency of our proposed method.

3. 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. We summarize the previous study by Martin et. al. [18] that derives threshold voltage, power consumption, and the performance of the design as functions of its supply and bias voltages. Subsequently, we proceed to present the power and performance parameters of the proposed 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 processor has 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.

ABB is a simple-to-implement technique to control subthreshold leakage. As the reverse body bias voltage is applied to the chip, the subthreshold leakage current will be reduced. Figure 2 shows a rough sketch of ABB implementation for a die. Note that the voltage regulators are physically on chip, and can be controlled via software instructions. The bias voltage 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. This scheme, changes the substrate (body) voltage for all of design transistors at once, and affects the subthreshold leakage current of each of them.

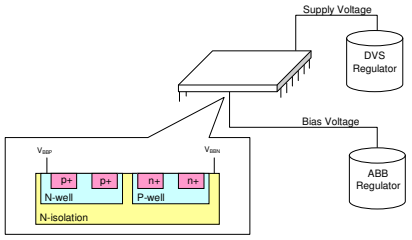

Figure 2: Adaptive Body Biasing Implementation

3.2 Threshold Voltage

The threshold voltage of a short-channel MOSFET transistor is given by [16]:

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

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. 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 [18]. 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 power consumption P_s can be approximated as:

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

where I_{sub_n} is the subthreshold leakage current, and I_{jn} and I_{bn} are the drain and source to body junction leakage currents in the NMOS device [12, 13, 17]. 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:

$$P = C_{eff} V_{dd}^2 f + V_{dd} K_3 e^{k_4 V_{dd}} e^{K_5 V_{bs}} + |V_{bs}| I_j \quad (5)$$

Where, K_i's are technology dependent constants [18]

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 internal 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 [8]:

$$t_{inverter} = \frac{L_d K_6}{(V_{dd} - V_{th})^\alpha} \quad (6)$$

where L_d is the logic depth of the path [17], 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:

$$f = (L_d K_6)^{-1} ((1 + K_1) V_{dd} + K_2 V_{bs} - V_{th_1})^\alpha \quad (7)$$

Equation (7) 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 supply voltage 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. Martin et. al. proposed an equation that can be used to find the energy optimal settings of supply and body bias for a given clock frequency [18].

4. DVS+ABB ENABLED PROCESSOR MODEL

We target a 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 the 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 Equation 7, there are infinitely many (supply voltage, body bias voltage) pairs that can cause the processor to operate at a given frequency. We utilize the equations (8) and (9) derived in [18] to find the energy optimal supply and body bias voltages that result in a given frequency and process technology. Equation (8) illustrates the relationship between the bias voltage and the derivative of the energy consumption per cycle. Equation (9) formulates the supply voltage as the function of the bias voltage.

$$\frac{\partial E_{cyc}}{\partial v_{bs}} = \begin{cases} L_g K_3 f^{-1} \left(K_1 V_{bs} + K_2 \right) e^{K_3 V_{bs} + K_4} - I_j L_g f^{-1} \\ \quad + 2 C_{eff} \left(K_5 V_{bs} + K_6 \right) \qquad if\ V_{dd} > 0.5 \\ \\ \frac{L_g}{2f} \left(K_3 K_5 e^{K_5 V_{bs} + 0.5 K_4} - 2 I_j \right) \quad otherwise \end{cases} \tag{8}$$

$$V_{dd} = \left(L_d K_6 f - K_2 V_{bs} + V_{th1} \right) / \left(1 + K_1 \right) \tag{9}$$

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.

We assume that our target processor can operate at 5 different clock frequencies, from $200MHz$ up to $1GHz$ at $200MHz$ steps. We adopt the process technology and processor parameters from Predictive Technology Models [1] and existing DVS-enabled commercial processors [2], respectively. Using Equations (8) and (9), we obtain the energy optimal supply and body bias voltages corresponding to each frequency. Table 1 demonstrates the characteristics of the operating modes for our target processor in 90nm.

Operating frequency(MHz)	1000	800	600	400	200
Supply voltage(V)	1.63	1.47	1.29	1.11	0.95
Bias voltage(V)	-0.08	-0.17	-0.25	-0.35	-0.47

Table 1: Processor operating modes at 90nm

5. LEAKAGE-AWARE COMPILATION

Our compiler optimization goal is to minimize the application total energy consumption by assigning different basic blocks of the code to different processor operating modes (or simply modes) subject to meeting the deadline of the application. We also consider the energy and latency penalty of switching modes. We aim to perform mode switching, by insertion of mode switch instructions on the control flow edges of the application. Since there might be several control flow edges that arrive at a common destination, 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. A similar approach has been used by Xie et al. [6] 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.

Figure 3 shows a partial view of a program control flow structure, which illustrates sample branches, loops, and self-loops among the basic blocks. We analyze the structure of the application and profile the typical execution traces of the application to extract the required statistical 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. We utilize the extracted information to perform mode assignment to execution traces of the application, under deadline constraint. We proceed to formulate this problem as an mixed-integer linear programming (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 are inserted to the code.

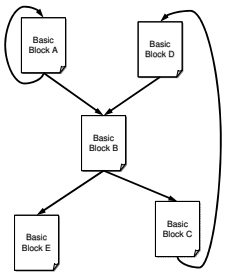

Figure 3: Partial CDFG structure of an example program

Upon switching the operating mode of the processor, the supply and body bias voltage regulators have to charge the capacitance. The charging of capacitance incurs some energy penalty, which can be estimated as:

$$E_s = (V_{dd_1}^2 - V_{dd_2}^2) C_r + (V_{bs_1}^2 - V_{bs_2}^2) C_s \tag{10}$$

where C_r is the capacitance of the power trail and C_s is the capacitance of the substrate of the device. V_{dd} and V_{bs} denote the supply and body bias voltages, respectively [20].

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 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 exited through edge (b,c) (Figure 3). 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 will run in the same operating mode. The 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:

$$\sum_{b=1}^{R}\sum_{c=1}^{R}\sum_{m=1}^{N} G_{bc}k_{bcm}E_{cm} + \sum_{a=1}^{R}\sum_{b=1}^{R}\sum_{c=1}^{R} D_{abc}\left|\sum_{m=1}^{N}(k_{abm}V_{m,s}^2 - k_{bcm}V_{m,s}^2)\right| C_r + \sum_{a=1}^{R}\sum_{b=1}^{R}\sum_{c=1}^{R} D_{abc}\left|\sum_{m=1}^{N}(k_{abm}V_{m,b}^2 - k_{bcm}V_{m,b}^2)\right| C_s \quad (11)$$

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 Equation (11) represents the energy consumption for execution of the basic blocks at their associated operating modes. The second and third terms are the switching energy penalty caused by DVS and ABB, respectively. Hence, the objective of our optimization is to minimize Equation (11) by manipulating the operating variables k_{bcm}.

The latency of the application is the sum total of the execution latency of basic blocks in all iterations and the delay penalty associated with mode switches. The switching delay can be approximated as the time required to charge the capacitance by voltage regulators [20]:

$$E_s = (V_{dd_1} - V_{dd_2})\frac{2C_r}{2I_{r,max}} + (V_{bs_1} - V_{bs_2})\frac{2C_s}{2I_{s,max}} \quad (12)$$

where $I_{r,max}$ and $I_{s,max}$ are the maximum possible currents. The execution deadline constraint of the application can be written as the following inequality:

$$\sum_{b=1}^{R}\sum_{c=1}^{R}\sum_{m=1}^{N} G_{bc}k_{bcm}T_{cm} + Max\left[\sum_{a=1}^{R}\sum_{b=1}^{R}\sum_{c=1}^{R} D_{abc}\left|\sum_{m=1}^{N}(k_{abm}V_{m,s} - k_{bcm}V_{m,s})\right| C_R , \sum_{a=1}^{R}\sum_{b=1}^{R}\sum_{c=1}^{R} D_{abc}\left|\sum_{m=1}^{N}(k_{abm}V_{m,b} - k_{bcm}V_{m,b})\right| C_B\right] \leq deadline \quad (13)$$

where C_R and C_B are constants equal to $\frac{2C_r}{I_{r,max}}$ and $\frac{2C_s}{I_{s,max}}$, respectively. T_{cm} is the average latency of basic block c under mode m, and $V_{m,s}$ and $V_{m,b}$ are constants representing the supply and body bias voltages in mode m. The optimization object of MILP formulation is to minimize (11), which quantifies the energy consumption of the compiled application. For most of embedded applications, the compiled code has to guarantee an execution deadline. We model this constraint by (13), which would prevent the MILP solver to generate solutions that violate the deadline constraint. Another MILP constraint is that the summation of the mode variables on each edge should be exactly 1 to guarantee a unique assignment of each edge to an operating mode:

$$\sum_{m=1}^{N} k_{bcm} = 1 \quad (14)$$

The absolute and max function used in equations (11) and (13) is not an MILP problem. To linearize the max function in the equation (13), we define a new variable which is greater than both elements inside the max function. We introduce variable e_{abc} to linearize the terms corresponding to supply voltage in equations (11). Three other variables and corresponding constraints are also added, to linearize the terms associated with supply voltage in (13), and with body bias voltage in (11) and (13). The corresponding constraints are removed for brevity.

$$-e_{abc} \leq \sum_{m=1}^{N}(k_{abm}V_{m,s}^2 - k_{bcm}V_{m,s}^2) \leq e_{abc} \quad (15)$$

6. EXPERIMENTAL RESULTS

6.1 Experimental Setup

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 4 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 read by 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.

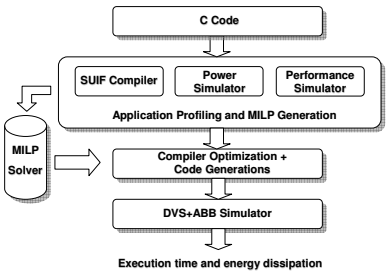

Figure 4: The setup of experiments for leakage-aware compilation

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 Equations 11 and 13), and execution frequency of control flow edges with respect to preceding active basic block (G_{bc} and D_{abc} in Equation 11). The front end of our compiler is based on the MachineSUIF compiler framework [14, 11]. We utilized MachineSUIF to generate CDFG representation of a program, and extract profiling information that relate to execution frequency of basic blocks.

Furthermore, we need to know 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 have modified the simplescalar simulator [19] to report the average latency of each basic block. A modified XTREM simulator [7] is used to report average energy dissipation of basic blocks. This methodology gives us cycle-accurate estimation of basic block latency and energy dissipation. Once the necessary profiling information are gathered, we generated the MILP instance. The CPLEX package is used to solve the problem instances.

We have selected five applications from MiBench[10] embedded application suite as our testbenches. The selected applications represent several application domains of embedded systems, including networking, automotive, telecom and security. Table 2 reports the characteristics of the selected applications. The selected application domains justify the need for deadline constraint and realtime operation of the generated code.

Benchmark	Application domain	# basic blocks	# lines of code	Dynamic instr. count (billion)
susan(corners)	automotive	203	2122	18.92
patricia	network	138	343	23.81
dijkstra	network	36	174	14.11
adpcm(coder)	telecom	33	281	9.35
sha	security	32	242	9.02

Table 2: Selected testbenches and their characteristic

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 5 visualizes the relative location of deadlines in comparison to baseline implementations. 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 $1GHz$ and $800MHz$ 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 $200MHz$.

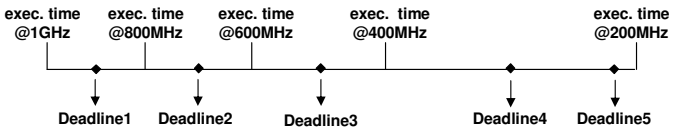

Figure 5: Relation of deadlines to baseline execution times

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 CFGs. However, it is possible that two consecutive mode switch instructions select the same operating mode, and hence, the second instruction redundant. Figure 6 shows examples of the redundant operating mode assignment. In the examples, basic block C will always be executed under the operating mode of basic block B. Therefore we can remove the mode switch basic block on the edge between basic block B and C. The mode switch basic block on the edge from E to D is also considered 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 that can be optimized using standard compiler optimization techniques. We apply standard optimization techniques to remove redundant mode switches, and improve the performance and energy dissipation of the generated code.

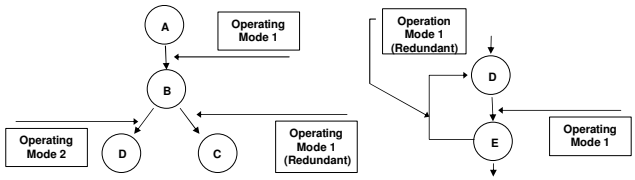

Figure 6: Examples of the redundant operating mode assignments

We have modified XTREM [7], a power simulator for Intel XScale DVS-enabled core, to estimate the energy dissipation of the generated code on the processor with 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, two pairs of supply and body bias voltages are used. For a given frequency, the DVS-enabled processor has zero body bias, and hence, its supply voltage is different from the DVS+ABB enabled processor. The optimal supply and bias voltage for each frequency are obtained using Equations (8) and (9). The developed simulator also considers the energy and delay penalty of switching between modes.

6.2 Experimental Result

We implemented the experimental flow depicted in Figure 4 and generated code for five applications listed in Table 2. To examine the impact of execution deadline and technology scaling, each application testbench is experimented under twenty different settings, i.e., five different execution execution deadlines (Figure 5) and four different process technologies. The models and parameters of the process technology are obtained from [1]. The reported execution time and energy dissipation are the numbers estimate from the generated code, using cycle-accurate simulators.

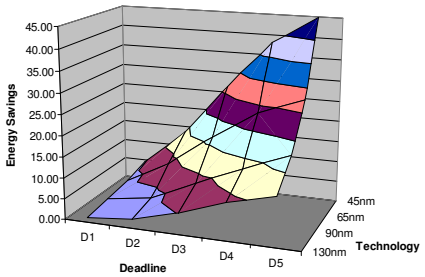

Figure 7: Average improvements in energy savings: DVS+ABB compared to baseline execution

Table 3 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 $800MHz$ 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.

Benchmark	Exec. time @1GHz	Exec. time @800MHz	Exec. time @600MHz	Exec. time @400MHz	Exec. time @200MHz	Average MILP solution time
adpcm	12.45	17.27	22.31	33.46	62.77	5.10
sha	11.57	14.08	20.42	28.72	58.89	5.59
dijkstra	17.4	21.17	32.54	42.3	84.63	5.22
susan	23.45	31.36	43.14	62.47	121.45	1588
patricia	28.43	38.01	52.14	74.61	146.21	183

Table 3: Applications execution time and MILP solution time (sec)

Due to the page limitation, we use two figures to show the energy impact of our method. Figure 7 illustrates the three dimensional space of average energy savings over benchmark applications, 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. Similarly, Figure 8 compares the energy dissipation of DVS+ABB with conventional DVS, over various deadline and process technologies.

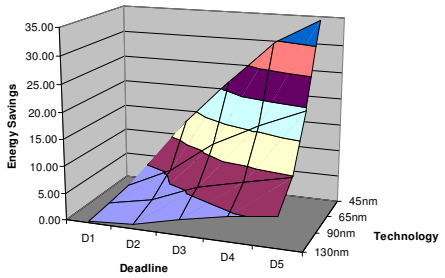

Figure 8: Average improvements in energy savings: DVS+ABB compared to conventional DVS

As figures 7 and 8 suggest, the energy savings significantly increase with the relaxation of deadline, for a given process technology. Intuitively, deadline relaxation increases the timing slack, which is better exploited by frequency scaling methods. For example in 65nm process, gradual relaxation of the timing constraint from deadline 1 to deadline 5 leads to 2.94%, 8.52%, 16.48%, 23.36% and 27.45% improvement in energy dissipation over baseline execution.

On the other hand, DVS+ABB is aware of the leakage's contribution to total dissipated energy for each technology node. Thus, 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 130nm, where, leakage energy is negligible. However with the shrinkage of the device sizes, the leakage energy increases exponentially. As a result, the energy dissipation gap between the two methods grows with technology scaling. For example considering deadline 4, our approach achieves 4.5%, 7.8%, 15.9% and 29.6% energy improvement over conventional DVS, for 130nm, 90nm, 65m and 45m, respectively.

7. 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 OS. 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 the energy dissipation gap between leakage-aware and conventional DVS grows with technology scaling. Future works include coordination of dynamic and static leakage-aware voltage scaling techniques.

8. REFERENCES

[1] http://www-device.eecs.berkeley.edu/ ptm/introduction.html.

[2] http://www.intel.com/design/intelxscale.

[3] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum. "Profile-Based Dynamic Voltage Scheduling Using Program Checkpoints". In *Design Automation and Test in Europe*, pages 168–175, March 2002.

[4] C.-H. Hsu, U. Kremer. "The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction". In *Conference on Programming Language Design and Implementation*, pages 38–48, June 2003.

[5] D. Duarte, N. Vijaykrishnan, M. J. Irwin, H.-S. Kim, G. McFarland. "Impact of scaling on the effectiveness of dynamic power reduction schemes". In *Proceeding of International Conference on Computer Design*, pages 382–387, September 2002.

[6] F. Xie,M. Martonosi,S. Malik. "Intraprogram Dynamic Voltage Scaling:Bounding Opportunities with Analytic Modeling". *ACM Transactions on Architecture and Code Optimization*, 1(3):1–45, September 2004.

[7] G. Contreras, M. Martonosi, J. Peng, R. Ju, G.Y. Lueh. "XTREM: a power simulator for the Intel XScale core". *ACM SIGPLAN Notices*, 39(7):115–125, July 2004.

[8] K.A. Bowman, B.L. Austin, J.C. Eble, X. Tang, J.D. Meindl. "A physical alpha-power law MOSFET model". *IEEE Journal of Solid-State Circuits*, 34(10):1410–1414, October 1999.

[9] L. Yan, J. Luo, N.K. Jha. "Combined dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems". In *IEEE/ACM International Conference on Computer-Aided Design*, pages 30–37, November 2003.

[10] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, and T. Mudge. "Mibench: a free, commercially representative embedded benchmark suite". In *Proceeding of the IEEE 4th Annual Workshop on Workload Characterization*, pages 3–14, December 2001.

[11] 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.

[12] M.J. Chen, H.T. Huang, C.S. Hou, K.N. Yang. "Back-gate bias enhanced band-to-band tunneling leakage in scaled MOSFETS". *IEEE Electron Device Letters*, 19(4):134–136, April 1998.

[13] M.R. Stan. "Optimal Voltages and Sizing for Low Power". In *Intl. VLSI Design Conf*, page 428, 1999.

[14] M.W. Hall, J.M. Anderson, S.P. Amarasinghe, B.R. Murphy, L. Shih-Wei, E. Bugnion, and M.S. Lam. "Maximizing Multiprocessor Performance with the SUIF Compiler". *Computer*, 29(12):84–89, 1996.

[15] N. AbouGhazaleh, D. Mossé, B.R. Childers, R. G. Melhem, M. Craven. "Collaborative Operating System and Compiler Power Management for Real-Time Applications". In *IEEE Real Time Technology and Applications Symposium*, pages 133–143, 2003.

[16] P. KO, J. Huang, Z. Liu, C. Hu. "BSIM3 for Analog and Digital Circuit Simulation". In *IEEE Symposium on VLSI Technology CAD*, pages 400–429, January 1993.

[17] R. Gonzales, B.M. Gordon, M.A. Horowitz. "Supply and Threshold Voltage Scaling for Low Power CMOS". *Journal of Solid-State Circuits*, 32(8):1210–1216, August 1997.

[18] S.M. Martin, K. Flautner, T. Mudge, D. Blaauw. "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads". In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 721–725, 2002.

[19] T. Austin, E. Larson, D. Ernst. "SimpleScalar: an infrastructure for computer system modeling". *Computer*, 35(2):59–67, February 2002.

[20] T.D. Burd, T.A. Pering, A.J. Stratakos, R.W. Brodersen. "A dynamic voltage scaled microprocessor system". *IEEE Journal of Solid-State Circuits*, 35(11):1571–1580, November 2000.