Real-Time Loop Scheduling with Energy Optimization via DVS and ABB for Multi-Core Embedded System

Guochen Hua[1], Meng Wang[1], Zili Shao[1], Hui Liu[2] and Chun Jason Xue[3]

[1] Department of Computing, The Hong Kong Polytechnic University, Hong Kong, {04994029d, csmewang, cszlshao}@comp.polyu.edu.hk
[2] Software Engineering Institute, Xidian University, Xi'an, China, liuhui@xidian.edu.cn
[3] City University of Hong Kong, Kowloon, Hong Kong, jasonxue@cityu.edu.hk

Abstract. Dynamic Voltage Scaling (DVS) is an effective technique to reduce energy consumption of processors by dynamically adjusting the operational frequency and supply voltage. However, with feature sizes shrinking, the achievable power saving by DVS is becoming limited as the leakage power increases exponentially. Adaptive Body Biasing (ABB) is an effective technique to reduce leakage power by increasing the circuit's threshold voltage via body biasing. In this paper, we propose a novel real-time loop scheduling technique to minimize both dynamic and leakage energy consumption via DVS and ABB for applications with loops considering voltage transition overhead. The proposed algorithm, EOLSDA (Energy Optimization Loop Scheduling with DVS and ABB), is designed to repeatedly regroup a loop based on rotation scheduling [4, 5] and decrease the energy consumption via DVS and ABB within a timing constraint. We conduct experiments on a set of DSP benchmarks based on the power model of 70nm technology. The results show that our technique achieves big energy saving compared with list scheduling [8] and the algorithm in [11].

1 Introduction

Power consumption has become an extremely important issue for real-time embedded systems due to its significant impact on battery life, system density, cooling cost, and system reliability, etc. Traditionally, dynamic power is the primary contributor to total power consumption. Dynamic Voltage Scaling (DVS) is one of the most effective techniques to reduce the dynamic power by dynamically scaling down the supply voltage and operational frequency. However, supply voltage scaling often requires a reduction in the threshold voltage, which leads to an exponential rise in the sub-threshold leakage current, and hence the leakage power consumption. As technology feature size continues to shrink, leakage power is becoming comparable to dynamic power in the current generation of technology, and it will dominate the overall energy consumption in future

technologies. Therefore, with the trend of adopting multi-cores in embedded systems, it is important to reduce both dynamic and leakage energy for multi-core embedded systems. As loops are prevalent in multimedia processing and Digital Signal Processing (DSP) applications, we focus on minimizing both dynamic and leakage energy consumption via DVS and ABB for real-time applications with loops considering time and energy transition overhead.

Many researches have been done on energy optimization using DVS and ABB for real-time embedded systems [9, 11, 1, 5, 2, 13, 6]. In [9], Martin et al. proposed a technique combining DVS and ABB to minimize both dynamic and leakage power consumption for unrelated tasks. And the expression for obtaining the optimal tradeoff between supply voltage and bias voltage is derived in that work. Yan et al. [13] proposed an algorithm with DVS and ABB for a task graph with real-time constraints. Huang et al. [6] proposed a leakage-aware compilation methodology that targets embedded processors with both DVS and ABB capabilities. Although these work can effectively reduce energy consumption, loop optimization is not considered. There has been some work on minimizing energy for application with loops. Saputra et al. [10] used several classic loop-oriented compiler optimization techniques such as loop fusion to reduce the execution time and then applied DVS to reduce energy. However, the whole loop is scaled with the same voltage in their work. Shao et al. [11] proposed a Dynamic Voltage Loop Scheduling algorithm (DVLS) to minimize the energy consumption considering transition overhead for applications with loops on multi-core embedded systems. However, leakage power is not considered in their work.

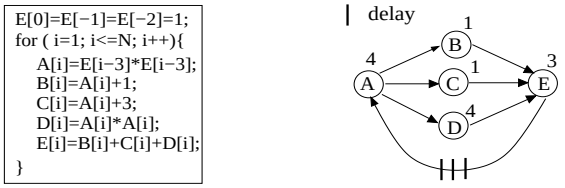
In this paper, we propose a novel real-time loop scheduling algorithm, EOLSDA (Energy Optimization Loop Scheduling with DVS and ABB), to minimize energy consumption via performing DVS and ABB simultaneously based on the power model [9] for applications with loops considering transition overhead. Our basic idea is to repeatedly regroup a loop based on rotation scheduling [4, 5] and decrease the energy consumption via DVS and ABB within a timing constraint. We conduct experiments on a set of DSP benchmarks based on a 70nm processor. The results show that our technique achieves big energy saving compared with list scheduling [8] and the algorithm in [11]. On average, our algorithm contributes to 65.61% energy reduction over list scheduling and 42.74% over DVLS algorithm [11]

The rest of the paper is organized as follows. The motivational examples are shown in Section 2. The models and basic concepts are introduced in Section 3. The EOLSDA algorithm is presented in Section 4. The experimental results and analysis are provided in Section 5, and the conclusion is given in Section 6.

2 Motivational Examples

In this section, we motivate the loop scheduling problem with energy minimization via DVS and ABB by showing how to schedule a cyclic DFG that represents a loop on a real-time multi-core embedded system. We compare the energy con-

sumption of the schedules generated by the list scheduling algorithm, the DVLS algorithm [11], and our technique.



(a) (b)

Node	Num of clock cycles	Dynamic Voltage Scaling				Combined DVS and ABB			
		High Voltage		Low Voltage		High Voltage		Low Voltage	
		Time	Energy	Time	Energy	Time	Energy	Time	Energy
A	4	0.4	10.48	0.8	6.72	0.4	5.88	0.8	3.04
B	1	0.1	2.62	0.2	1.68	0.1	1.47	0.2	0.76
C	1	0.1	2.62	0.2	1.68	0.1	1.47	0.2	0.76
D	4	0.4	10.48	0.8	6.72	0.4	5.88	0.8	3.04
E	3	0.3	7.86	0.6	5.04	0.3	4.41	0.6	2.28

(c)

Fig. 1. (a) A loop application. (b) The corresponding DFG (Data Flow Graph). (c) The number of clock cycles, execution time, and the energy of each node.

A loop application is shown in Figure 1(a). The corresponding DFG is shown in Figure 1(b). In DFG, each node represents the computation task in the loop, the edge without delay represents the intra-iteration data dependency (e.g. $A \rightarrow B$), the edge with delays represents the inter-iteration data dependency (e.g. $E \rightarrow A$ has three delays which are denoted by three bars), the number of delays represents the number of iterations involved, and the number beside each node represents the execution time of that node.

Assume that there are two processor cores in the multi-core embedded system, and the maximum frequency of each processor core is 10 GHz with the supply voltage/body bias voltage pair of (0.7244V, 0V). Note that the body bias voltage is set to 0 since there is no body bias voltage applied here. According to the power model introduced in [9], we get $P = 26.2\mu W$, $T = 0.1ns$, where P and T represent the power consumption and clock period, respectively. There are two voltage scaling approaches for reducing power consumption. One is Dynamic Voltage Scaling, and the other is combined Dynamic Voltage Scaling and Adaptive Body Biasing.

We first consider applying the Dynamic Voltage Scaling. Assume that the operational frequency can be scaled to 50% of the maximum frequency. By performing DVS, the frequency and the supply voltage can be scaled down from 10 GHz to 5 GHz, and 0.7244V to 0.4770V, respectively. Thus, the power consumption is reduced from $26.2\mu W$ to $8.4\mu W$.

Then we consider the approach that combines Dynamic Voltage Scaling and Adaptive Body Biasing. In this approach, we first apply the body bias voltage to the maximum frequency. Based on the power model in [9], we can obtain the optimal pair of supply and body bias voltage, which is (0.8189V, -0.6566V). Therefore, we get the power consumption reduced from $26.2\mu W$ to $14.7\mu W$. Similarly, if the frequency is scaled down by 50%, by obtaining the optimal pair of supply and body bias voltage, which is (0.5795V, -0.7124V), the power consumption can be reduced to $3.8\mu W$. The execution time and energy of each node via DVS alone and combined DVS and ABB are shown in 1(c), where the time unit is ns and the energy unit is $10^{-15}J$. Since the voltage transition causes both time and energy overhead, we assume it takes $0.1ns$ with $10^{-15}J$ to transit between different frequency levels. These assumptions are only for demonstration purpose. Our technique is general enough to deal with general energy models as discussed in later sections.

Assume that the timing constraint is $1.2ns$ which is the upper bound for the schedule length of the loop. If the execution time of a loop iteration is less than a given timing constraint, we use the real execution time for energy calculation.

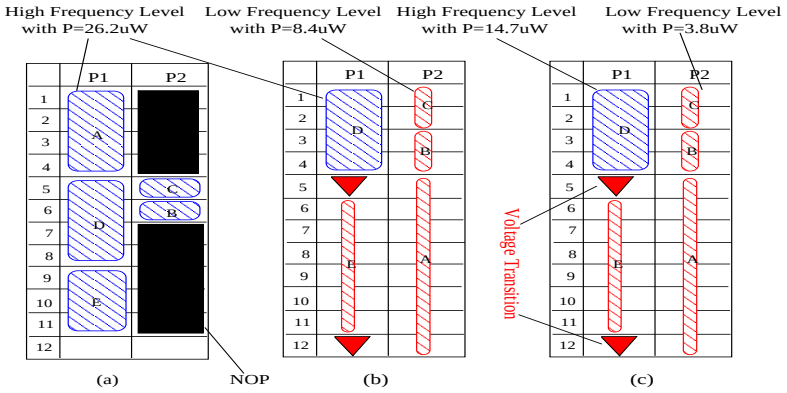


Fig. 2. The schedules generated by list scheduling, DVLS and our technique, respectively. The energy: (a) $2 * 1.1 * 26.2 = 57.64 \times 10^{-15}J$, (b) $26.2 * 0.4 + 2 + 8.4 * 1.8 = 27.6 \times 10^{-15}J$, and (c) $14.7 * 0.4 + 2 + 3.8 * 1.8 = 14.72 \times 10^{-15}J$.

In the following, we compare the energy consumption of the schedules generated by different techniques. Note that each time slot in the schedule represents 0.1ns. We obtain the first schedule by the list scheduling (shown in Figure 2(a)), where both processor cores operate at the maximum frequency level for the best timing performance. Consider all empty slots filled with NOP operations, we get the energy consumption $E = 2 * 1.1 * P_{max} = 2 * 1.1 * 26.2 = 57.64 \times 10^{-15}J$.

The second schedule shown in Figure 2(b) is generated by the DVLS algorithm [11] which repeatedly rotates the schedule and applies DVS for energy minimization. Two frequency levels 10 GHz (the high level) and 5 GHz (the

low level) are used in this example, with $P_H = 26.2\mu W$, $P_L = 8.4\mu W$, and $T_H = 0.1ns$, $T_L = 0.2ns$, respectively. This schedule is generated after applying DVLS for 3 rotation steps based on the list schedule in Figure 2(a). In this schedule, nodes A, B, C, and E are assigned to the low level, while node D is assigned to the high level. Processor core P1 operates at two different frequency levels by processing D at the high level and E at the low level. Considering the transition overhead, the energy consumption of this schedule is $E = P_H*0.4+2*E_{tran}+P_L*(0.6+1.2) = 26.2*0.4+2+8.4*1.8 = 27.6\times10^{-15}J$.

The schedule generated by our EOLSDA algorithm is shown in Figure 2(c). Compared with the schedule in Figure 2(b), this schedule consumes less energy since leakage power is reduced as well as dynamic power by performing DVS and ABB simultaneously. In EOLSDA, we also use the two frequency levels 10 GHz and 5 GHz, with $T_H = 0.1ns$ and $P_H = 14.7\mu W$ for the high level, and $T_L = 0.2ns$ and $P_L = 3.8\mu W$ for the low level. The total energy consumption of this schedule is $E = P_H*0.4+2*Etran+P_L*(0.6+1.2) = 14.7*0.4+2+3.8*1.8 = 14.72 \times 10^{-15}J$. The results show that our EOLSDA algorithm achieves big energy saving compared with the list scheduling and DVLS algorithm.

3 Models and Concepts

In this section, we introduce the power model and some basic concepts that will be used in this paper.

3.1 Power Model

In this subsection, we summarize the previous power model [9] that derives power consumption and the performance as functions of the supply and body bias voltages.

Power Consumption In MOSFET circuit, there are three major sources of power consumption: dynamic power, static power, and short circuit power. The short circuit power consumption occurs only during signal transitions and is negligible [12]. The dynamic power, P_{AC} is given by,

$$P_{AC} = C_{eff}V_{dd}^2 f \tag{1}$$

where C_{eff} is the average switched capacitance per cycle, and f is the clock frequency. The static power consumption consists of the power due to sub-threshold leakage current and reverse bias junction current. Thus, the static power consumption, P_{DC} is given by,

$$P_{DC} = V_{dd}I_{subn} + |V_{bs}|\,(I_{jn} + I_{bn}) \tag{2}$$

where I_{subn} is the sub-threshold leakage current, I_{jn} and I_{bn} are the drain to body junction leakage current and source to body junction leakage current in

the NMOS device. As $I_{jn} + I_{bn}$ can be approximated as a constant, Ij , the static power can be expressed as,

$$P_{DC} = V_{dd} K_3 e^{K_4 V_{dd}} e^{K_5 V_{bs}} + |V_{bs}| I_j \tag{3}$$

thus the total power consumption, P, becomes,

$$P = C_{eff} V_{dd}^2 f + V_{dd} K_3 e^{K_4 V_{dd}} e^{K_5 V_{bs}} + |V_{bs}| I_j \tag{4}$$

therefore, the total energy consumed per cycle is given by,

$$E_{cyc} = C_{eff} V_{dd}^2 + L_g f^{-1} (V_{dd} K_3 e^{K_4 V_{dd}} e^{K_5 V_{bs}} + |V_{bs}| I_j) \tag{5}$$

Energy Optimization To optimize the energy consumption, we utilize the equation (8) and (9) derived in [9] to find the optimal pairs of supply and body bias voltages for given frequencies and process technologies. Equation (8) illustrates the relationship between the body bias voltage and the derivative of the total energy consumption per cycle. Equation (9) formulates the supply voltage as a function of the body bias voltage.

$$\frac{\partial E_{cyc}}{\partial V_{bs}} = \{ L_g K_3 f^{-1} (k_1 V_{bs} + k_2) e^{k_3 V_{bs} + k_4} - I_j L_g f^{-1} \qquad + 2 C_{eff} (k_5 V_{bs} + k_6) \tag{6}$$

$$V_{dd} = (L_d K_6 f - K_2 V_{bs} + V_{th1}) / (1 + K_1) \tag{7}$$

3.2 Rotation Scheduling

Rotation Scheduling [3, 4] is a scheduling technique used to optimize a loop schedule with resource constraints. It transforms a schedule to a more compact one iteratively in a DFG. By using rotation scheduling, we can get more opportunities to reschedule nodes of DFG to better locations so that the length of a schedule can be reduced. In Figure 3, we show an example to explain how to obtain a new schedule via rotation scheduling. Using the schedule generated by list scheduling in Figure 2(a) as an initial schedule, we rotate the nodes at the first row of the schedule down. The rotated graph is shown in Figure 3(a), the new schedule is shown in Figure 3(b), and the equivalent loop body after rotation is shown in Figure 3(c).

From the program point of view, rotation scheduling regroups a loop body and attempts to reduce intra-dependencies among nodes. In this case, after the rotation, a new loop is obtained by the transformation as shown in Figure 3(c), where the corresponding computation for node A is rotated and put at the end of the new loop body. One iteration from the old loop is separated and put outside the new loop body: the computation for node A is put in the prologue and those for the other nodes are put in the epilogue. In the new loop body, node A performs the computation for the $(i+1)_{th}$ iteration when the other nodes do the computation of the i_{th}. The transformed loop body after the rotation scheduling can be obtained based on the retiming values of nodes [4]. The code

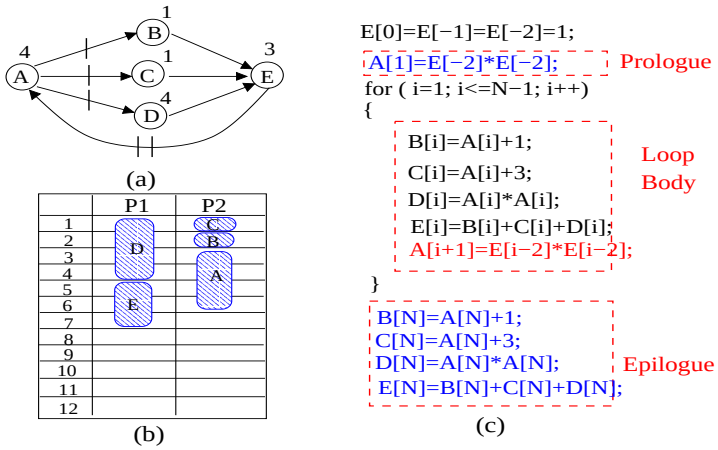


Fig. 3. (a) The rotated DFG. (b) The schedule after the rotation. (c) The equivalent loop after regrouping loop body.

size is increased by introducing the prologue and epilogue after the rotation is performed. This problem can be solved by the code size reduction technique proposed in [14].

In the new schedule, the schedule length is reduced from 11 to 7 after the first rotation. Therefore, more opportunities are provided for energy optimization by DVS and ABB. In the following, we introduce our EOLSDA (Energy Optimization Loop Scheduling with DVS and ABB) technique that can effectively reduce energy with loop optimization and combined DVS and ABB.

4 The Energy Optimization Loop Scheduling with DVS and ABB Algorithm

In this section, we propose our real-time loop scheduling algorithm, EOLSDA (Energy Optimization Loop Scheduling with DVS and ABB), to minimize energy consumption via DVS and ABB for applications with loops. Our basic idea is to repeatedly regroup a loop based on rotation scheduling [3, 4] and decrease the energy by combined DVS and ABB technique as much as possible within a timing constraint. The EOLSDA algorithm is shown in Algorithm **??**. In EOLSDA, based on an initial schedule, we repeatedly reschedule nodes and adjust their frequency levels for energy minimization. The initial schedule can be obtained by assigning each node with the maximum frequency level using the list scheduling.

In the EOLSDA algorithm, we first put all rotatable nodes into Rotate_Node_Set and do retiming. If a node is the first node scheduled on a processor and there is at least one delay in each of its incoming edge, then the node is rotatable. For a node u that is not the first node scheduled on a processor, it is rotatable if the following two conditions are satisfied:

1. There is at least one delay in each of its incoming edge, and
2. All nodes scheduled before u on the same processor are rotatable.

In this way, we can rotate all nodes without disobeying dependencies [4].

After the rotatable set is obtained, the nodes in it are ordered based on the execution time so the node with longer execution time will be rescheduled earlier. Following such order, we can obtain a schedule that can satisfy the timing constraint as much as possible. The reason is that the frequency level of a rotated node will be adjusted for energy minimization in the next step so its execution time may be increased. Therefore, we should first reschedule the node with the longest execution time to avoid the situation that we can not find an empty slot with enough length to put it in.

When rescheduling a node, we try to find the earliest empty slot to put the node in, following the dependencies in the retimed graph. After finding this empty slot, we first put the node into it and adjust its frequency level so the minimum energy consumption can be obtained for the empty slot. This can be achieved by trying all possible frequency levels of the node and filling the rest of the empty slot with all frequency levels of NOP operations. Then we calculate the total energy of the empty slot and select the frequency level of the node with the minimum energy consumption.

It is possible that no empty slots can be found to put the rotated node in. In this case, we assign the node with the highest frequency level so that the node has the minimum execution time. We then repeat the above rescheduling steps to try to put the node into an empty slot with the earliest schedule time and adjust the frequency level for energy minimization. If we still can not find an empty slot even with the highest frequency level, we will pick up a processor to put the node to the end of it in such a way that the node is rescheduled as the last node on the processor with the earliest schedule time. Finally, we calculate the energy of the schedule and record the schedule with the minimum energy consumption if the timing constraint is satisfied with the schedule.

Figure 4 shows an example. Given the DFG shown in Figure 1(b) and the initial schedule in Figure 2(a), the schedules generated by the EOLSDA algorithm in three consecutive rotations are shown in Figure 4(a)-(c), respectively. As shown in the example, in each rotation, all rotated nodes are put into the rotation set and rescheduled with frequency adjustment for energy minimization. After three rotations, the schedule shown in Figure 4(c) achieves the minimum energy consumption for the DFG.

In the following, we perform complexity analysis on the EOLSDA algorithm. Let P_Num be the number of processor cores and k the number of the available frequency levels. Let $|V|$ and $|E|$ be the node and edge numbers of a given graph, respectively. For the EOLSDA algorithm, let R_Num be the number of rotations. In one rotation, we can finish the retiming in $O(|V||E|)$ and it takes at most $O(|V| * log|V|)$ to order the nodes in the rotation set. When doing rescheduling, we need to reschedule at most $|V|$ nodes and it takes $O(|E| + P\_Num * |V|)$ to find an empty slot with the earliest schedule time where it takes at most $O(P\_Num * |V|)$ to find all empty slots in a schedule and takes at most $O(|E|)$ to

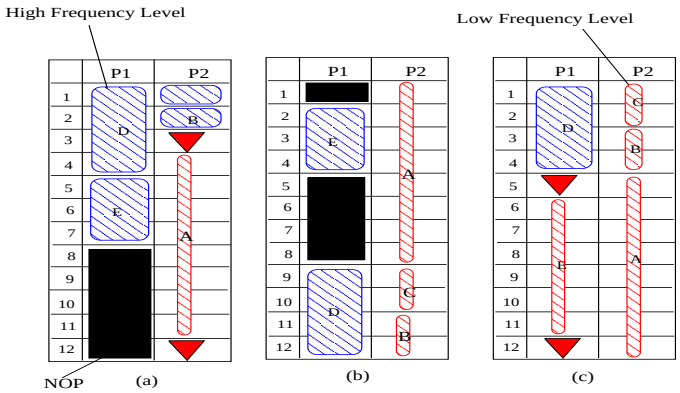


Fig. 4. The schedules generated by the EOLSDA algorithm for the DFG in Figure 1(b) with the initial schedule in Figure 2(a) after (a) the first rotation, (b) the second rotation, and (c) the third rotation.

determine the earliest available slot considering the edge dependencies. So totally it takes $O(|V|(|V|+|E|))$ to do rescheduling considering P_Num is a constant, and the EOLSDA algorithm can be finished in $O((|V|^2+|V||E|)*R\_Num)$.

5 Experiments

In this section, we do experiments with a set of benchmarks including Infinite Impulse Response filter (IIR), 8-stage lattice filter (8-Stage), the differential equation solver (DEQ), elliptic filter (Elliptic), and voltera filter (Voltera). In the experiments, we set the rotation times as $10*Node\_Num$ where $Node\_Num$ is the number of nodes in the DFG. The results show that the rotation times to generate the best schedule is less than $1*Node_Num$ and close to the times when all nodes have been rotated one time.

Frequencies f_{op} (GHz)	Supply Voltage V_{dd} (V)	Bias Voltage V_{bs} (V)	Power P (μW)
7.8	0.712	-0.667	8.76
10.4	0.839	-0.656	15.97
13	0.968	-0.661	26.28
15.6	1	-0.676	40.27

Table 1. The frequency levels, corresponding Supply voltage, Body Bias Voltage and Power based on the power model of a 70nm processor.

The experiments are conducted base on the power model of 70nm processor. We configure the parameters of the processor according to the power model in [9]. The frequency level of the processor can be varied between 50-100% in 16%

steps, with the maximum frequency of 15.6GHz. According to the power model, we can obtain the optimal supply and body bias voltage for a given frequency. Then, the energy consumption per cycle can be calculated by using equation (9), and the power is derived from the formula $E_{cyc} = P/f$. The four frequency levels, as well as their corresponding Vdd and Vbs and energy consumptions, are shown in Table 1. The time overhead during a voltage transition among the above four voltage levels is calculated based on equation (12), and the energy overhead is calculated based on equation (13).

We obtain the number of clock cycles for instructions from the IA-32 architecture manual [7]. Basically, a NOP instruction takes one clock cycle, an Addition instruction with memory operands takes three clock cycles, and a Multiplication instruction with memory operands takes six clock cycles. Base on the above latencies, the execution time of each node is small in terms of transition overhead. Therefore, an enlarge factor of 1000000 is applied to the execution time of each node while all data dependency relations are kept. Moreover, since the clock cycle is too small (for instance, 0.13ns), we set the unit of time slot for scheduling is 0.01ns. Thus, the instructions can be scheduled in integer number of time slots, e.g. an NOP instruction will take 13 time slot to execute at the frequency level of 7.8GHz.

Frequencies f_{op} (GHz)	Supply Voltage V_{dd} (V)	Power P (μW)
7.8	0.616	16.12
10.4	0.744	27.15
13	0.8728	43
15.6	1	65

Table 2. The frequency levels, corresponding Supply voltage and Power of a 70nm processor applying DVS only.

To evaluate the performance of our algorithm, we compare the energy consumption with the list scheduling and DVLS algorithm [11]. The energy consumption of the schedule generated by list scheduling is calculated under the maximum frequency level. For DVLS algorithm, we set all the $V_{bs} = 0$ since body bias voltage is not considered in [11]. In this case, for the same frequency scaling levels, we calculate the corresponding supply voltage and energy consumption for each frequency level according to equation (15) and (9). Table 2 lists the frequency levels of the 70nm processor when applying DVS.

For each benchmark, we apply 10 timing constraints with $0.5ns$ steps, and first timing constraint we use is the approximated minimum execution time from the list scheduling. The experiments are conducted on the systems with 2, 3 and 4 processor cores, respectively.

The experimental results are shown in Table 3. For each benchmark, we list the average energy consumption. In Table 3, column "TC" represents the range of timing constraints we applied. Sub-columns "Energy" under columns

Bench.	TC Range (ns)	List Energy (10^{-15}J)	DVLS Energy (10^{-15}J)	EOLSDA Energy (10^{-15}J)	EOLSDA Imp-List (%)	EOLSDA Imp-DVLS (%)
2 processor cores						
IIR	3-7.5	315.21	186.11	107.15	66.01	42.43
DEQ	2-6.5	229.11	138.49	80.47	64.88	41.89
ELF	5-9.5	547.16	390.73	223.48	59.16	42.80
Voltera	5-9.5	566.42	385.19	223.59	60.53	41.95
8-Stage	9-13.5	820.80	533.23	302.96	63.09	43.18
Average Imp. (2 Cores)					**62.73**	**42.45**
3 processor cores						
IIR	3-7.5	321.60	177.63	99.84	68.96	43.79
DEQ	2-6.5	236.19	131.47	76.27	67.71	41.98
ELF	5-9.5	548.49	360.46	202.57	63.07	43.80
Voltera	5-9.5	563.52	335.05	181.71	67.75	45.77
8-Stage	9-13.5	820.60	429.30	237.66	71.21	44.64
Average Imp. (3 Cores)					**67.74**	**44.00**
4 processor cores						
IIR	3-7.5	322.88	179.49	105.58	67.30	41.17
DEQ	2-6.5	246.80	138.21	82.04	66.76	40.64
ELF	5-9.5	554.44	384.20	221.26	60.09	42.41
Voltera	5-9.5	568.32	318.92	191.44	66.32	39.97
8-Stage	9-13.5	830.40	426.71	238.05	71.33	44.21
Average Imp. (4 Cores)					**66.36**	**41.68**
Total Average Improvement					**65.61**	**42.71**

Table 3. The energy comparison for the schedules generated by list scheduling, the DVLS algorithm and the EOLSDA algorithm.

"List", "DVLS" and "EOLSDA" represent the average energy obtained by the list scheduling, DVLS and EOLSDA, respectively. Sub-column "Imp-List" under column "EOLSDA" represents the energy reduction of the EOLSDA algorithm over list scheduling. Sub-column "Imp-DVLS" under column "EOLSDA" represents the energy reduction of the EOLSDA over DVLS. The total average energy reduction of EOLSDA is shown in the last row.

The results show that the EOLSDA algorithm achieves great energy reduction compared with list scheduling and DVLS. On average, EOLSDA has a energy reduction of 65.61% over list scheduling, and a reduction of 42.71% over DVLS.

6 Conclusion

In this paper, we proposed a novel real-time loop scheduling algorithm to minimize both dynamic and leakage power consumption via performing DVS and ABB simultaneously for applications with loops considering transition overhead. The proposed algorithm, EOLSDA, is designed to repeatedly regroup a loop

based on rotation scheduling [4] and decrease the energy by combining DVS and ABB as much as possible within a timing constraint. We conducted experiments on a set of DSP benchmarks based on the power model of the 70nm processor. The experimental results show that our algorithm achieves great energy reduction compared with list scheduling [8] and the algorithm in [11].

Acknowledgment

The work described in this paper was partially supported by the grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (PolyU A-PH13, PolyU A-PA5X, PolyU A-PH41, and PolyU B-Q06B).

References

1. A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. Al-Hashimi. Overhead-conscious voltage selection for dynamic and leakage power reduction of time-constraint systems. In *DATE04*, pages 518–523, 2004.
2. A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. Al-Hashimi. Simultaneous communication and processor voltage scaling for dynamic and leakage energy reduction in time-constrained systems. In *DATE04*, pages 362–369, 2004.
3. L.-F. Chao. *Scheduling and Behavioral Transformations for Parallel Systems.* PhD thesis, Princeton University, 1993.
4. L.-F. Chao, A. S. LaPaugh, and E. H.-M. Sha. Rotation scheduling: A loop pipelining algorithm. *TCAD*, 16(3):229–239, 1997.
5. Y. Chen, Z. Shao, Q. Zhuge, C. Xue, B. Xiao, and E. Sha. Minimizing energy via loop scheduling and dvs for multi-core embedded systems. *Parallel and Distributed Systems*, 2:2–6, July 2005.
6. P. Huang and S. Ghiasi. Power-aware compilation for embedded processors with dynamic voltage scaling and adaptive body biasing capabilities. In *DATE-06*, pages 943–944, 2006.
7. Intel. *IA-32 Intel Architecture Optimization Reference Manual*, April 2006.
8. D. Landskov, S. Davidson, B. Shriver, and P. W. Mallett. Local microcode compaction techniques. *ACM Computing Surveys*, 12(3):261–294, Sept 1980.
9. S. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *ICCAD-02*, pages 721–725, 2002.
10. H. Saputra and M. Kandemir. Energy-conscious compilation based on voltage sacling. In *LCTES'02*, 2002.
11. Z. Shao, M.Wang, Y. Chen, C. Xue, M. Qiu, L. T. Yang, and E. H.-M. Sha. Real-time dynamic voltage loop scheduling for multi-core embedded systems. *Accepted in IEEE Transactions on Circuits and Systems II (TCAS-II).*
12. H. Veendrick. Short-circuit dissipation of static cmos circuitry and its impact on the design of buffer circuits. *IEEE J. Solid-State Circuits*, 19:468–473, Aug 1984.
13. L. Yan, J. Luo, and N. K. Jha. Joint dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems. *TCAD*, 24(7):1030–1041, July 2005.
14. Q. Zhuge, B. Xiao, and E. H.-M. Sha. Code size reduction technique and implementation for software-pipelined dsp applications. *TECS*, 2(4):1–24, Nov. 2003.