Real-Time Dynamic Voltage Loop Scheduling for Multi-Core Embedded Systems

Zili Shao Meng Wang Ying Chen Chun Xue Meikang Qiu Laurence T. Yang Edwin H.-M. Sha

***Abstract*— In this paper, we propose a novel real-time loop scheduling technique to minimize energy consumption via DVS for applications with loops considering transition overhead. Two algorithms, IDVS and DVLS, are designed integrating with DVS. IDVS is an algorithm to iteratively optimize the DAG (Directed Acyclic Graph) part of a loop by incorporating transition overhead into optimization scheduling scheme. To the best of our knowledge, it is the first work in voltage scheduling for the model with dependent tasks on multi-processor systems considering voltage transition overhead. DVLS is an algorithm to repeatedly regroup a loop based on rotation scheduling [1] and decrease the energy by DVS as much as possible within a timing constraint. We conduct the experiments on a set of DSP benchmarks based on the power model of AMD Mobile Athlon 4 DVS processors. The experimental results show that our algorithms achieve big energy saving compared with the traditional time-performance-oriented scheduling algorithm.**

***Index Terms*— DVS, real-time, multi-core, embedded systems, scheduling, loop.**

I. Introduction

Low power is extremely important for real-time embedded systems. DVS (Dynamic Voltage Scaling) is one of the most powerful techniques to reduce energy consumption by adjusting supply voltage at running time. Using mode-set instructions, the supply voltage can be changed to different voltage modes, which makes it possible to implement DVS by software. With the trend of more multi-cores being used in embedded systems, it is important to study DVS for multi-core systems. Since loops are prevalent in multimedia processing and Digital Signal Processing (DSP) applications, this paper focuses on minimizing energy via DVS for real-time applications with loops considering practical limitations of DVS processors. Both time and energy overhead during a voltage transition is considered and analyzed in this work.

Many researches have been done on energy optimizations using DVS for embedded systems during resent years ([2], [3], [4], [5], [6], [7]). In [3], Zhang et al. discussed inter-task scheduling based on ILP (Integer Linear Programming). Shin et al. [2] focused on intra-task DVS for real-time applications based on static timing analysis. In recent work, the DVS scheduling combining with communication overhead [6] and leakage power [7] has been studied. Although these work can effectively reduce energy consumption by DVS, loop optimization is not considered. Transition overhead is also ignored in the above work. There has been some work on minimizing energy for applications with loops using DVS. In [8], Saputra et al. 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. In this paper, we develop a dynamic voltage loop scheduling technique that can leverage the voltage level of each task node.

Zili Shao and Meng Wang are with the Dept. of Computing at the Hong Kong Polytechnic Univ.; Ying Chen is with the Dept. of CSE at the Shanghai Jiao-Tong Univ.; Chun Xue, Meikang Qiu and Edwin H.-M. Sha are with the Dept. of Computer Science at the Univ. of Texas at Dallas; Laurence T. Yang is with Dept. of Computer Science at St. Francis Xavier University.

In this paper, we propose a novel real-time loop scheduling technique to minimize energy consumption via DVS for applications with loops considering transition overhead. Two voltage scheduling algorithms, IDVS (Iterative Dynamic Voltage Scheduling) and DVLS (Dynamic Voltage Loop Scheduling), are designed integrating with DVS. IDVS is an algorithm to iteratively optimize the DAG (Directed Acyclic Graph) part of a loop by incorporating transition overhead into optimization scheduling scheme. To the best of our knowledge, it is the first work in voltage scheduling for the model with dependent tasks on multi-processor systems considering voltage transition overhead. DVLS is an algorithm to repeatedly regroup a loop based on rotation scheduling [1] and decrease the energy by DVS as much as possible within a timing constraint. Both algorithms can be used in design space exploration, real-time operating systems or compilers to optimize energy consumption. We conduct experiments on a set of DSP benchmarks based on the power model of AMD Mobile Athlon 4 DVS processors [9]. The experimental results show that our algorithms achieve big energy saving compared with the traditional time-performance-oriented scheduling algorithm. The second algorithm is the best and can greatly reduce energy consumption.

The rest of the paper is organized as following: In Section II, we give motivational examples. The models and basic concepts are introduced in Section III. In Section IV, we propose our algorithms. The experimental results are shown in Section V, and the conclusion is shown in Section VI.

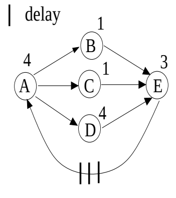
II. Motivational Examples

In this section, we motivate the dynamic voltage loop scheduling problem by showing how to schedule a cyclic DFG (Data Flow Graph) that represents a loop on a multi-core embedded systems. We compare the energy consumption of the schedules generated by the list scheduling algorithm, the algorithm in [3], and our technique.

A loop application is shown in Figure 1(a). The corresponding DFG (Data-Flow Graph) is shown in Figure 1(b), where 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 number of clock cycles needed. In the paper, intra-iteration dependencies mean the dependencies inside an iteration while inter-iteration dependencies mean the dependencies among different iterations.

```
E[0]=E[−1]=E[−2]=1;
for ( i=1; i<=N; i++){
   A[i]=E[i−3]*E[i−3];
   B[i]=A[i]+1;
   C[i]=A[i]+3;
   D[i]=A[i]*A[i];
   E[i]=B[i]+C[i]+D[i];
}
```

(a)



(b)

Node	Num of clock cycles	High Voltage		Low Voltage	
		Time	Energy	Time	Energy
A	4	4	12	8	4
B	1	1	4	2	1
C	1	1	4	2	1
D	4	4	16	8	4
E	3	3	12	6	3

(c)

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

Assume that there are two processor cores in this multi-core embedded system, and each DVS processor has two voltage/frequency levels, the high level and low level. Based on the power model,

$P = C_{SW} f_{op} V_{dd}^2$ [10], without loss of generality, we assume that $C_{SW} = 1nF$; the voltage/frequency pair is (2 V, 1 GHz) at the high level and (1 V, 0.5 GHz) at the low level. Therefore, we get $P_H = 4W$, $P_L = 0.5W$, $T_H = 1ns$, $T_L = 2ns$, where P_H and P_L are used to represent the high-level and low-level power consumptions, respectively, and T_H and T_L the high-level and low-level clock periods, respectively. The number of clock cycles of a node is not changed with DVS. Thus, we get the execution time and energy of each node in Figure 1(c), where the time unit is ns and the energy unit is nJ. A voltage transition causes both time and energy overhead. Assume that it takes 1 ns with 1 nJ to transit between different voltage/frequency levels. These assumptions are only for demonstration purpose. Our technique is general enough to deal general energy models as discussed in later sections.

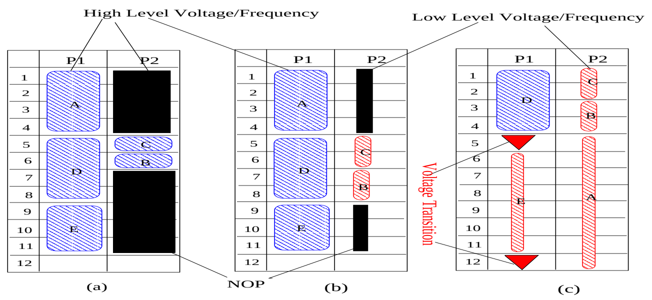


Fig. 2. (a) The schedule by list scheduling with the energy 4*11*2=88 nJ. (b) The schedule by ILP-Scheduling from [3] with the energy 4*11+0.5*11=49.5 nJ. (c) The schedule by our technique with the energy 4*4+1+0.5*6+1+0.5*12=27 nJ.

Assume that the timing constraint is 12 ns. In this paper, a timing constraint is the upper bound we can use for executing one iteration of a loop. If the execution time of a loop iteration is less than a given timing constraint, we will use the real execution time for energy calculation. In the following, we compare the energy consumption of the schedules generated by different techniques. We obtain the first schedule by the list scheduling (shown in Figure 2(a)), where both processor cores operate at the high voltage levels for the best timing performance. Considering all empty slots filled with NOP operations, we get the energy consumption $E = 2 * P_H * 11 = 2 * 4 * 11 = 88nJ$.

The second schedule shown in Fig. 2(b) is obtained by the DVS technique in [3], in which it uses a 2-phase voltage assignment and scheduling approach and achieves the near-optimal solution. This technique does not consider transition overhead but it works well for this example because there is no transition in this case. From the schedule, we can see that node B and C can be processed with the low voltage level. Therefore, the processor core, P2, can operate at the low voltage level. The energy consumption is $E = P_H * 11 + P_L * 11 = 4 * 11 + 0.5 * 11 = 49.5nJ$, which is the optimal for this example by optimizing the DAG part of the loop.

The schedule generated by our DVLS algorithm in Section IV is shown in Fig. 2(c). In DVLS, we perform the loop optimization with DVS. As shown in Fig. 3, the loop body is regrouped (Figure 3(c)) and the dependencies in the loop are changed (Figure 3(a)) using loop optimization. Therefore, we can change voltage levels of more nodes and reduce more energy consumption. This schedule is generated after we apply DVLS for 3 rotation steps based on the schedule shown in Fig. 2(a). In the schedule, nodes A, B, C and E are assigned to the low voltage. The processor core, P1, operates at two different voltage levels by processing D at the high level and E at the low level. Considering the transition overhead, we get the energy $E = P_H * 4 + 2 * E_{Transit} + P_L * (6 + 12) = 4 * 4 + 2 * 1 + 0.5 * 18 = 27nJ$. Based on the schedule obtained by our DVLS algorithm, the tasks can be scheduled with the insertion of voltage-setting instructions. The technique can be integrated into task schedulers in real-time operating systems or compilers to generate energy-efficient code.

III. Models and Concepts

In this section, we introduce the system model and some basic concepts that will be used in the later sections.

A. The Models

We focus on real time applications on multi-core embedded systems. An application with loops is represented by a DFG (Data Flow Graph). There is a timing constraint and it must be satisfied when each iteration of a loop is executed. In a multi-core system, there are multiple processor cores, and each processor core can operate at a finite set of supply voltage levels $Voltage\_Level = \{V_{dd_1}, V_{dd_2}, \cdots, V_{dd_k}\}$. The voltage level of a processor core can be changed independently by voltage-level-setting instructions without influencing other cores. The power consumption of a processor core at a voltage level is calculated by $P = C_{SW} f_{op} V_{dd}^2$ [10], where C_{SW} is the capacitance and f_{op} is the frequency.

It causes both time and energy overhead during a voltage transition. From [11], for a voltage change from $V_{DD_1} \rightarrow V_{DD_2}$, the time transition, t_{TRAN}, can be calculated by:

$$t_{TRAN} = \frac{2 * C_{DD}}{I_{MAX}} |V_{DD_2} - V_{DD_1}| \qquad (1)$$

where C_{DD} is the capacitance of the voltage converter and I_{MAX} is the maximum output current of the converter. The energy transition, E_{TRAN}, consists of two parts, $E_{TRAN-DC}$ and $E_{TRAN-CPU}$ [11], [12]. The energy consumed by the voltage converter, $E_{TRAN-DC}$, is:

$$E_{TRAN-DC} = \alpha * C_{DD} * |V_{DD_1}^2 - V_{DD_2}^2| \qquad (2)$$

where α is the efficiency factor of voltage converter. The energy consumed by the CPU during the transition, $E_{TRAN-CPU}$, is:

$$E_{TRAN-CPU} = P * t_{TRAN} \qquad (3)$$

where P is the power consumption at the voltage level entered in the transition. The above overhead model is applied for the normal voltage transition. For the transition involving sleep states, the time transition is quite large considering the synchronization delay with off-chip components such as memory. The power consumption and transition time at sleep states can be obtained from data sheets of a processor. Then we can calculate the energy overhead based on Equations 2 and 3.

B. Cyclic DFG and Static Schedules

We use a cyclic Data Flow Graph (DFG) to denote a loop in our works. A cyclic DFG G=⟨V, E_SET, d, N⟩is a node-weighted and edge-weighted directed graph, where V is a set of nodes and each node denotes a computation task in the loop, E_SET is the edge set, d(e) is a function to represent the number of delays for any edge $e \in$ E_SET, N(u) is a function to represent the computation cycles for any node $u \in V$. The edge without delay represents the intra-iteration data dependency; the edge with delays represents the inter-iteration data dependency and the number of delays represents the number of iterations involved. In this paper, intra-iteration dependencies mean the dependencies inside an iteration while inter-iteration dependencies mean the dependencies among different iterations.

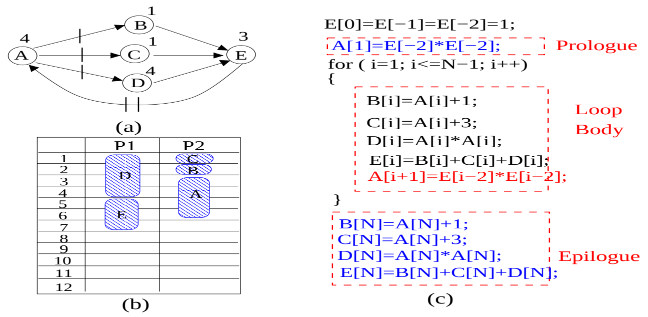
Using the loop and DFG shown in Figure 1(a) and (b) as an example, $V = \{A, B, C, D, E\}$ is the node set, and E_SET= $\{A \rightarrow B,\ A \rightarrow C,\ A \rightarrow D,\ B \rightarrow E,\ C \rightarrow E,\ D \rightarrow E,\ E \rightarrow A\}$ is the edge set. $N(A) = 4$ denotes that the computation cycles of node A is 4. d($A \rightarrow B$)=0 means that there is no delay in edge $A \rightarrow B$ so the edge represents the intra-iteration data dependency. d($E \rightarrow A$)=3 means that there are three delays in edge $E \rightarrow A$ so the edge represents the inter-iteration data dependency and the data generated by node E three iterations ago are needed for processing node A at the current iteration.

From the DFG of an application, we can obtain a static schedule. A static schedule of a cyclic DFG is a repeated pattern of an execution of the corresponding loop. In our works, a schedule implies both control step assignment and allocation. A static schedule must obey the dependency relations of the Directed Acyclic Graph (DAG) portion of the DFG. The DAG is obtained by removing all edges with delays in the DFG. From the DFG shown in Figure 1(b), we obtain a static schedule as shown in Figure 2(a) generated by list scheduling.

C. Retiming and Rotation Scheduling

Retiming [13] is an optimal scheduling technique for cyclic DFGs considering inter-iteration dependencies. It can be used to optimize the cycle period of a cyclic DFG by evenly distributing the delays. Retiming generates the optimal schedule for a cyclic DFG when there is no resource constraint. Rotation Scheduling [1] 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. In most cases, the minimal schedule length can be obtained in polynomial time by rotation scheduling.

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).



```
E[0]=E[−1]=E[−2]=1;
A[1]=E[−2]*E[−2];
for ( i=1; i<=N−1; i++)
{
    B[i]=A[i]+1;
    C[i]=A[i]+3;
    D[i]=A[i]*A[i];
    E[i]=B[i]+C[i]+D[i];
    A[i+1]=E[i−2]*E[i−2];
}
B[N]=A[N]+1;
C[N]=A[N]+3;
D[N]=A[N]*A[N];
E[N]=B[N]+C[N]+D[N];
```

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

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 [1]. The code 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 10 to 7 after the first rotation. Therefore, more opportunities are provided for energy optimization by DVS. In the following, we introduce our dynamic voltage loop scheduling technique that can effectively reduce energy with loop optimization and DVS.

IV. The Dynamic Voltage Scheduling Algorithms

In this section, we propose our dynamic voltage scheduling algorithms. Two algorithms, IDVS (Iterative Dynamic Voltage Scheduling) and DVLS (Dynamic Voltage Loop Scheduling), are proposed to produce schedules with minimum energy for real-time applications with loops. IDVS is an algorithm to iteratively optimize the DAG (Directed Acyclic Graph) part of a loop by incorporating transition overhead into optimization scheduling scheme. To our best knowledge, it is the first work in voltage scheduling for the model with dependent tasks on multi-processor systems considering voltage transition overhead. DVLS is an algorithm to repeatedly regroup a loop based on rotation scheduling [1] and decrease the energy by DVS as much as possible within a timing constraint. In the following, we first propose the IDVS and DVLS algorithms in Sections IV-A and IV-B, respectively, and then perform complexity analysis in Section IV-C.

A. The IDVS Algorithm

In the IDVS (Iterative Dynamic Voltage Scheduling) algorithm, our basic idea is first to assign each node with the minimum voltage level and then iteratively change the voltage levels of nodes to reduce execution time until the timing constraint is satisfied. The transition overhead is incorporated into the optimization scheduling scheme during the voltage change.

Basically, the IDVS algorithm has two steps. In the first step, we assign each node with the minimum voltage level and then obtain an initial schedule by list scheduling, in which we set the priority of each node as the longest path from this node to a leaf node. Because the execution time of each node is the longest with the minimum voltage level, the schedule length may be too long and the timing constraint may not be satisfied. Therefore, in next step, we iteratively change voltage levels of nodes to reduce the execution times of nodes until the timing constraint is satisfied.

When changing voltage levels of nodes, in each iteration, we pick up one node with an unchanged voltage level and do voltage change, in which the maximum time reduction can be obtained among all nodes with all unchanged voltage levels. Since the nodes on the critical path determine the schedule length, we will select a node from the critical path to do voltage-level change in order to obtain the maximum schedule length reduction. Therefore, for each node u on the critical path, with all its unchanged voltage levels, we calculate the possible reduction time by:

$$Reduce\_Time(u) = Time\_Current\_VL(u) - \\ Time\_Changed\_VL(u) - Transition\_Time$$

where $Time\_Current\_VL(u)$ and $Time\_Changed\_VL(u)$ are the execution times of u with the current voltage level and the changed voltage level, respectively, and $Transition\_Time$ is the transition time from the changed voltage level to the voltage level of the node scheduled exactly below node u on the same processor core in the schedule. Then the node with the maximum $Reduce\_Time$ is selected and its voltage level is changed accordingly. For the nodes with the same $Redcue\_Time$, the node with earliest schedule time is selected. In this way, we can provide the best opportunity for schedule length reduction. With this voltage change, we keep the schedule sequence of nodes in each processor core and do rescheduling in such a way that the time reduction obtained in the previous step can play the biggest role for minimizing schedule length (the detailed algorithm is shown in in Attachment 1).

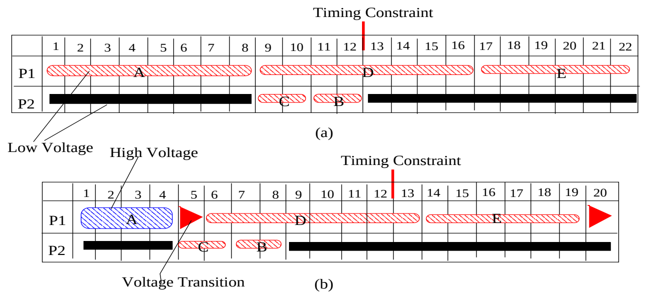


Fig. 4. The schedules generated by the IDVS algorithm after the fi rst voltage change: (a) The initial schedule by assigning each node with the minimum voltage level. (b) The schedule after the voltage change by changing the voltage of A from low to high.

Fig. 4 shows an example. For the DFG in Figures 1(b), the schedules generated by the IDVS algorithm are shown in Fig. 4. The initial schedule is shown in Fig. 4(a) that is obtained using list scheduling by assigning each node with the minimum voltage level. In the algorithm, we calculate Reduce_Time(u) for each node u on the critical path by considering all its unchanged voltage levels. The critical path is $A \rightarrow D \rightarrow E$. For example, for Node A, if we change the voltage from low to high, we obtain Reduce_Time(A)=8-4-1, where 8 is the execution time with the current (low) voltage, 4 is the time with the changed (high) voltage, and 1 is the transition time considering that the voltage level of D, the node scheduled exactly below A on P1, is low. After calculating all nodes on the critical path with all unchanged voltage levels, we find that we can obtain the maximum time reduction by changing the voltage of A. Therefore, its voltage is changed to the low level and we do rescheduling by keeping the node sequence on each processor. The schedule generated is shown in Fig. 4(b). After three iterations, we can get the schedule shown in Fig. 2(b) that achieves the minimum energy consumption in terms of the graph obtained by only considering the DAG part of the DFG. Our IDVS algorithm is general and can be applied to solve the voltage scheduling problem with transition overhead.

B. The DVLS Algorithm

In the IDVS algorithm, we perform optimization based on the DAG part of a DFG and do not consider inter-iteration dependencies. As shown in Section III, by exploring inter-iteration dependencies, we can get more opportunities to reschedule nodes of DFG to better locations. Therefore, we propose another algorithm, the DVLS (Dynamic Voltage Loop Scheduling) algorithm in this section. The basic idea of the DVLS algorithm is to repeatedly regroup a loop based on rotation scheduling [1] and decrease the energy by DVS as much as possible within a timing constraint.

In the DVLS algorithm, based on an initial schedule, we repeatedly reschedule nodes and adjust their voltage levels for energy minimization. The initial schedule can be obtained by assigning each node with the maximum voltage level using the list scheduling, which is done in our experiments. In the 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 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 [1].

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

When rescheduling a node, we try to put it into an empty slot in such a way that the node can be scheduled earliest. Then we adjust its voltage level so the minimum energy consumption can be obtained. This can be achieved by trying all possible voltage levels of the node with all voltage levels of NOP operations to fill the empty slot, and picking up the voltage level that causes the minimum energy.

It is possible, however, that we can not find any empty slot to put the rotated node in. In this case, we assign the node with the highest voltage level so 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 voltage level for energy minimization. If we still can not find an empty slot even with the highest voltage 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 (the detailed algorithm is shown in in Attachment 2).

Fig. 5 shows an example. Given the DFG shown in Fig. 1(b) and the initial schedule in Fig. 2(a), the schedules generated by the DVLS algorithms in three consecutive rotations are shown in Fig. 5(a)-(c), respectively. As shown in the example, in each rotation, all rotated nodes are put into the rotation set and rescheduled with voltage adjustment for energy minimization. After three rotations, the schedule shown in Fig. 5(c) achieves the minimum energy consumption for the DFG. Compared with the schedule generated by the IDVS algorithm, it achieves 45.5% energy improvment as shown in Fig. 2.

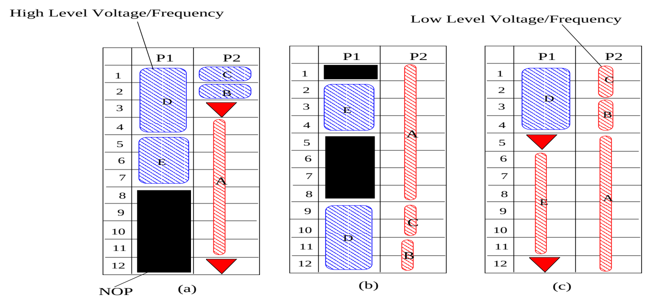


Fig. 5. The schedules generated by the DVLS algorithm for the DFG in Fig. 1(b) with the initial schedule in Fig. 2 after (a) the first rotation, (b) the second rotation, and (c) the third rotation.

C. Complexity Analysis

Let P_Num be the number of processor cores and k the number of the available voltage levels. Let $|V|$ and $|E|$ be the node and edge numbers of a given graph, respectively. In the IDVS algorithm, list scheduling takes $O(|V||E|P\_Num)$ to get an initial schedule. And in each voltage change iteration, it takes $O(k|V|)$ to find the node with the voltage change and $O(|V||E|P\_Num)$ to do rescheduling. The iteration will be repeated at most $k|V|$ times. So the IDVS algorithm can be finished in $O(|V|^2|E|)$ considering k and P_Num are constants.

For the IDVS 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 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 IDVS algorithm can be finished in $O((|V|^2+|V||E|)*R\_Num)$.

V. EXPERIMENTS

In this section, we experiment with our algorithms on a set of benchmarks including Infinite Impulse Response filter (IIR), 4-stage lattice filter (4-Stage), 8-stage lattice filter (8-Stage), the differential equation solver (DEQ), elliptic filter (Elliptic) and voltera filter(Voltera). In the experiments, the number of rotations is $10*Node\_Num$ where $Node\_Num$ is the number of nodes in a graph. The experimental results show that the number of rotations to generate the best schedule is less than $1*Node\_Num$ and close to the times when all nodes have been rotated one time.

Voltages V_{dd} (V)	Frequencies f_{op} (MHz)	Power P (W)	Voltages V_{dd} (V)	Frequencies f_{op} (MHz)	Power P (W)
1.2	500	9.2	1.25	600	12.0
1.3	700	15.1	1.35	800	18.6
1.4	1000	25.0			

TABLE I
THE VOLTAGE LEVELS, FREQUENCIES AND POWER BASED ON THE POWER MODEL OF THE MOBILE ATHLON4 PROCESSOR [9].

The experiments are conducted based on the power model of the AMD Mobile Athlon4 DVS processor [9]. The AMD Mobile Athlon4 processor can operate at various voltage levels in the range of 1.2-1.4 V with 50 mV steps, and the corresponding frequencies vary from 500 MHz to 1 GHz with 100 MHz steps[12]. The power is calculated by $P = C_{SW} f_{op} V_{dd}^2$ [10], where C_{SW} is 12.75 nF from the data sheet of the AMD Mobile Athlon4 processor [9]. The five voltage levels, their corresponding frequencies and power are shown in Table I.

The time overhead during a voltage transition among the above five voltage levels is calculated based on Equation 1, $t_{TRAN} = \frac{2*C_{DD}}{I_{MAX}}|V_{DD_2} - V_{DD_1}|$, in which C_{DD} and I_{MAX} are set as 5 μF and 1 mA based on an optimized converter from [11]. The energy overhead is calculated based on Equations 2 and 3. For the energy consumed by the converter in Equation 2, $E_{TRAN-DC} = \alpha * C_{DD} * |V_{DD_1}^2 - V_{DD_2}^2|$, α is set as 0.9 [11].

AMD Mobile Athlon 4 DVS processors have low power sleep states that can be utilized when systems are idle. The power consumed in the sleep state is 2.4 W [9]. For the transition involving sleep states, considering the synchronization delay with off-chip components such as memory, the transition time is quite large. For example, it takes 20 ms to synchronize with the main memory entering in or exiting from the sleep state for Compaq iPAQ [15]. Therefore, in our experiments, a sleep state with 2.4 W is used, and the time transition to enter in or exit from the sleep state is set as 20 ms.

The number of clock cycles for instructions are obtained from the IA-32 architecture manual [16] since the IA-32 instruction set is used by AMD Mobile Athlon 4 DVS processors. Basically, an 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. Based on the above latencies, however, the execution time of each node is too small in terms of the transition overhead. So we could not test our algorithms with transition overhead if we directly apply the latencies. In the experiments, therefore, the execution time of each node is enlarged by 2000 times while all data dependency relations are kept. In this way, we can test the effectiveness of our algorithms for optimizing the energy consumption by DVS with transition overhead. In the experiments, we only consider the loop kernel for each benchmark, and we do not do any other changes for program sizes of the benchmarks.

Bench.	TC	List	IDVS		DVLS		
	Range (μs)	Energy (μJ)	Energy (μJ)	Imp-IL (%)	Energy (μJ)	Imp-DL (%)	Imp-DI (%)
2 processor cores							
IIR	80-170	4000.00	3543.31	11.42	3230.52	19.24	8.59
4-Stage	130-220	6500.00	6276.53	3.44	5747.61	11.58	8.48
8-Stage	200-290	10500.00	9795.93	6.71	9270.74	11.71	5.36
DEQ	60-150	3000.00	2475.61	17.48	2416.52	19.45	2.31
Elliptic	140-230	7000.00	6329.23	9.58	6139.15	12.30	3.05
Voltera	140-230	7500.00	6754.23	9.94	6122.86	18.36	9.27
Average Imp. (2 Processor Cores)				**9.76**	–	**15.44**	**6.18**
3 processor cores							
IIR	80-170	6000.00	3890.52	34.45	3628.72	39.52	6.63
4-Stage	130-220	9750.00	5677.84	41.77	5312.22	45.52	5.80
8-Stage	200-290	15000.00	9523.32	38.31	7977.22	46.82	12.67
DEQ	60-150	4500.00	3086.25	31.42	3003.78	33.25	3.24
Elliptic	140-230	10500.00	6791.89	35.32	6570.62	37.42	3.24
Voltera	140-230	10500.00	6469.30	38.39	5970.40	43.14	7.21
Average Imp. (3 Processor Cores)				**36.61**	–	**40.94**	**6.46**
Total Average Improvement				**23.19**	–	**28.19**	**6.32**

TABLE II

THE ENERGY COMPARISON FOR THE SCHEDULES GENERATED BY LIST SCHEDULING, THE IDVS ALGORITHM AND THE DVLS ALGORITHM.

From the previous work, we could not find any techniques that deal with dependent tasks on multi-processor systems considering voltage transition overhead. Therefore, we compare our two algorithms with the list scheduling in which all nodes are processed at the highest voltage level. For each benchmark, we apply 10 different timing constraints: The minimum execution time from the list scheduling is used first, and then we gradually increase the timing constraint by 10 μs each time. The experiments are conducted on the systems with 2 and 3 processor cores. The experimental results are shown in Table II. Because of the limited space, for each benchmark, we only list the average energy (the detailed experimental results can be found in Attachment 3).

In TableII, column "TC Range" represents timing constraints we used that start from the minimum execution time and increase by 10 μs each step. Columns "Energy" under columns "List", "IDVS" and "DVLS" represent the average energy obtained by the list scheduling, our IDVS algorithm, and our DVLS algorithm, respectively. Columns "Imp-IL" and "Imp-DL" represent the improvement of IDVS and DVLS over the the list scheduling, respectively. Column "Imp-DI" represents the improvement of DVLS over IDVS. The total average improvement of IDVS and DVLS is shown in the last row.

From the experimental results, we can see that our algorithms achieve more energy saving compared with the list scheduling. On average, IDVS shows a 23.19% reduction and DVLS shows a 28.19% reduction. DVLS achieves better energy saving compared with IDVS, and a 6.32% reduction is obtained on average.

VI. Conclusion and Work In Progress

In this paper, we proposed a novel real-time loop scheduling technique to minimize energy consumption via DVS for applications with loops considering transition overhead. Two algorithms, IDVS and DVLS, are designed integrating with DVS. IDVS is an algorithm to iteratively optimize the DAG (Directed Acyclic Graph) part of a loop by incorporating transition overhead into optimization scheduling scheme. DVLS is an algorithm to repeatedly regroup a loop based on rotation scheduling [1] and decrease the energy by DVS as much as possible within a timing constraint. We conducted the experiments on a set of DSP benchmarks based on the power model of AMD Mobile Athlon 4 DVS processors. The experimental results show that our algorithms achieve big energy saving compared with the traditional time-performance-oriented scheduling algorithm. We are currently working on extending our algorithms by considering communication overhead and leakage power.

References

[1] L.-F. Chao, A. S. LaPaugh, and E. H.-M. Sha, "Rotation scheduling: A loop pipelining algorithm," *TCAD*, vol. 16, no. 3, pp. 229–239, 1997.

[2] D. Shin, J. Kim, and S. Lee, "Low-energy intra-task voltage scheduling using static timing analysis," in *DAC*, 2001, pp. 438–443.

[3] Y. Zhang, X. Hu, and D. Z. Chen, "Task scheduling and voltage selection for energy minimization," in *DAC*, 2002, pp. 183–188.

[4] W. C. Kwon and T. Kim, "Optimal voltage allocation techniques for dynamically variable voltage processors," in *DAC*, 2003, pp. 125–130.

[5] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processor," in *ISLPED*, 1998, pp. 197 –202.

[6] J. Luo, N. K. Jha, and L.-S. Peh, "Simultaneous dynamic voltage scaling of processors and communication links in real-time distributed embedded systems," *Accept for publication in TVLSI*.

[7] L. Yan, J. Luo, and N. K. Jha, "Joint dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems," *TCAD*, vol. 24, no. 7, pp. 1030–1041, July 2005.

[8] H. Saputra, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, J. S. Hu, C.-H. Hsu, and U. Kremer, "Energy-conscious compilation based on voltage scaling," in *LCTES'02*, June 2002.

[9] *Mobile AMD Athlon 4 Processor Model 6 CPGA Data Sheet rev:e*, Advanced Micro Devices, Tech. Rep. 24319, Nov. 2001.

[10] A. Chandrakasan, S. Sheng, and R. W. Brodersen, *Low-Power Digital CMOS Design*. Norwell, MA: Kluwer, 1996.

[11] T. D. Burd, "Energy-efficient processor system design," Ph.D. dissertation, Univ. of California, Berkeley, 2000.

[12] B. C. Mochocki, X. S. Hu, and G. Quan, "A unified approach to variable voltage scheduling for noideal dvs processors," *TCAD*, vol. 23, no. 9, pp. 1370–1377, Sept 2004.

[13] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, pp. 5–35, 1991.

[14] Q. Zhuge, B. Xiao, and E. H.-M. Sha, "Code size reduction technique and implementation for software-pipelined dsp applications," *TECS*, vol. 2, no. 4, pp. 1–24, Nov. 2003.

[15] *Compaq IPAQ h360 Hardware design specification - Version 0.2f*, http://www.handhelds.org/Compaq/iPAQH3600/iPAQ_3600.html [Online].

[16] *IA-32 Intel Architecture Optimization Reference Manual*, Intel, April 2006.