Minimizing Energy via Loop Scheduling and DVS for Multi-Core Embedded Systems

Ying Chen[1] Zili Shao[2] Qingfeng Zhuge[2] Chun Xue[2] Bin Xiao[3] Edwin H.-M. Sha[1,2]
[1]Dept. of CSE, Shanghai Jiao-Tong Univ., Shanghai, China
[2]Department of CS, Univ. of Texas at Dallas, Richardson, TX, USA
[3]Department of Computing, Hong Kong Polytechnic Univ., Hung Hom, Kowloon, Hong Kong

Abstract

Low energy consumptions are extremely important in real-time embedded systems, and scheduling is one of the techniques used to obtain lower energy consumptions. In this paper, we propose loop scheduling algorithms for minimizing energy based on rotation scheduling and DVS (Dynamic Voltage and frequency Scaling) for real-time loop applications. The experimental results show that our algorithms have better performances than list scheduling and pure ILP (Integer Linear Programming) scheduling with DVS.

1. Introduction

Low power and low energy consumptions are extremely important fr real-time embedded systems. Dynamic voltage and frequency scaling (DVS) is one of the most effective techniques to reduce energy consumption. In many microprocessor systems, the supply voltage can be change by mode-set instructions, which makes it possible to implement DVS by software. With the trend of multiple cores being widely used in embedded systems, it is important to study DVS techniques for multi-core embedded systems. Since applications with loops are prevalent in multimedia processing, digital signal processing, etc., this paper focuses on minimizing energy via loop scheduling and DVS for real-time loop applications.

Many researches have been done on DVS for real-time applications in recent years [5, 6, 4]. Zhang et. al. [6] proposed an ILP (Integer Linear Programming) model to solve DVS on multiple processor systems. Shin et. al. [5] proposed a DVS technique for real-time applications based on static timing analysis. However, in the above work, applications are modeled as DAG (Directed Acyclic Graph), and loop optimization is not considered. Saputra et. al. [4] considered loop optimization with DVS. However, in their work, the whole loop is scaled with the same voltage. Our technique can leverage the voltage of each task node.

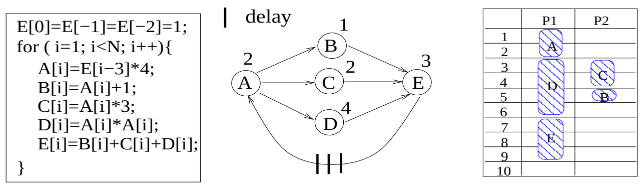
Integrating with DVS, we design new loop scheduling algorithms for real-time applications that produce schedules consuming minimal energy. In our algorithms, we use rotation scheduling [1] to get schedules for loop applications. The schedule length will be reduced after rotation. Then, we use DVS to assign voltages to computations individually in order to decrease the voltages of processors as much as possible within the timing constraint. The experimental data show that our algorithms can get better results on energy saving than the previous work. On average, BALOSA shows a 11.84% reduction and ILOSA shows a 15.68% reduction compared with the DVS technique in [6].

The rest of the paper is organized as following: In Section2, we give motivational examples. The models and basic concepts are introduced in Section 3. In Section 4, we propose our algorithms. The experimental results are shown in Section 5, and the conclusion is shown in Section 6.

2. Motivational Examples

Let's consider the effectiveness of rotation scheduling and DVS for a loop application shown in Figure 1(a). Figure 1(b) is the Data-Flow Graph (DFG) of the application, where nodes represent the computations, 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. The numbers beside nodes represent the computation cycles of nodes.

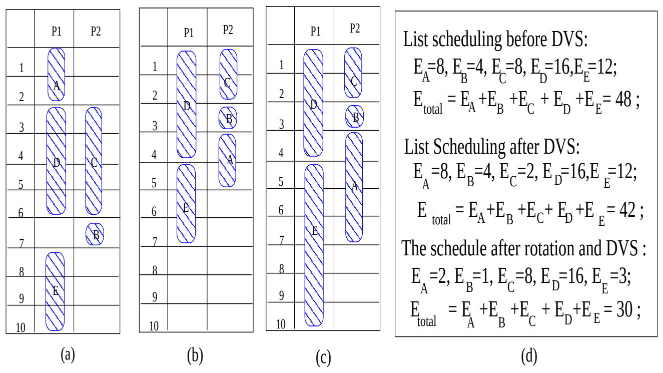
Assume that there are two processor cores in our multi-core system, there are two levels of voltage available, low and high, and the time constraint of this application is 10. According to the system model of DVS [5], the computation time is proportional to $V/(V-V_t)^2$, where V is the supply voltage, V_t is the threshold voltage; the energy consumption is proportional to V_{dd}^2. So here we assume the computation time of a node under the low voltage is twice as much as it is under the high voltage; the energy consump-



```
E[0]=E[−1]=E[−2]=1;
for ( i=1; i<N; i++){
    A[i]=E[i−3]*4;
    B[i]=A[i]+1;
    C[i]=A[i]*3;
    D[i]=A[i]*A[i];
    E[i]=B[i]+C[i]+D[i];
}
```

Figure 1. (a) A loop application. (b) The DFG. (c) The static schedule generated by list scheduling.

tion of a node under the high voltage is four times as much as it is under the low voltage. We get the first scheduling by list scheduling (shown in Figure 1(c)), where all computations are assigned the high voltage. The length of schedule is 9 (We will show the relative formulas in section 3). So for this schedule, we could do little on reducing the energy consumption, because the timing constraint of the application is 10. Figure 2(a) shows another schedule, where node C is assigned to the low voltage, and more space are utilized in the schedule (we call it relaxation).



List scheduling before DVS:

E_A=8, E_B=4, E_C=8, E_D=16,E_E=12;

$E_{total} = E_A + E_B + E_C + E_D + E_E = 48$;

List Scheduling after DVS:

E_A=8, E_B=4, E_C=2, E_D=16,E_E=12;

$E_{total} = E_A + E_B + E_C + E_D + E_E = 42$;

The schedule after rotation and DVS :

E_A=2, E_B=1, E_C=8, E_D=16, E_E=3;

$E_{total} = E_A + E_B + E_C + E_D + E_E = 30$;

Figure 2. (a) The original schedule after DVS. (b) The schedule after rotation. (c) The schedule after rotation and DVS. (d) The energy comparison of schedules.

By rotating the schedule shown in Figure 1(c), we can get a new scheduling (shown in Figure 2(b)). We notice that after rotation, the schedule length is reduced to 7. Now we can change the voltages for more nodes from high to low. Figure 2(c) shows the schedule where the nodes A, B, and E are assigned the low voltage. Figure 2(d) shows the energy consumption of three schedules shown in Figure 1(c), Figure 2(a) and Figure 2(c). The reduced energy via one rotation and DVS is considerable, which is 37.5%, compared to the schedule in Figure 1(c).

3. Models and Concepts

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

System Model: We focus on real time applications on multi-core embedded systems. An application with loops is represented by a DFG. There is a timing constraint and it must be satisfied for executing each iteration of a loop. In the multi-core system, there are multiple processor cores. Each processor core has multiple discrete levels of voltages and its voltage level can be changed independently by voltage-level-setting instructions without the influence for other cores.

Energy Model: Let $N(u)$ represent the number of computation cycles for node u, and $t(u)$ represent the execution time of u. We use the similar energy model as in [2, 5]. Given the number of cycles $N(u)$ of node u, the supply voltage V_{dd} and the threshold voltage V_{th}, its computation time t_u and the energy E_u are calculated as follows:

$$T_c = \frac{k \times V_{dd}}{(V_{dd} - V_{th})^{\alpha}} \qquad (1)$$

$$t_u = N(u) \times T_c = N(u) \times \frac{k \times V_{dd}}{(V_{dd} - V_{th})^2)^{\alpha}} \qquad (2)$$

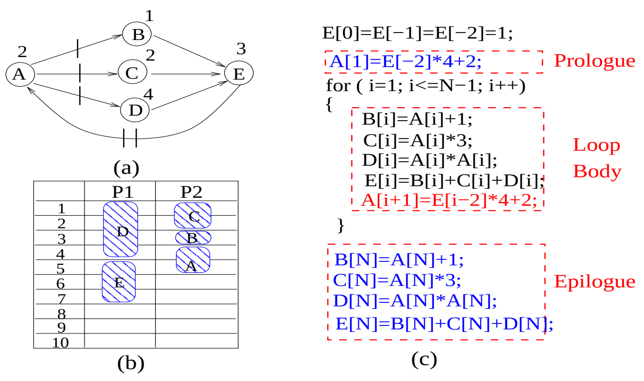
$$E_u = N(u) \times C_u \times V_{dd}^2 \qquad (3)$$

In Equation 1, T_c is the cycle period time, k is a device related parameter and α is a factor having the value from 2 to 1.2. In Equation 3, C_u is the effective switching capacitance per cycle. From Equations 2 and 3, we can see that the lower voltage will prolong the execution time of a node but reduces its energy consumption. In this paper, we assume that there is no energy or delay penalty associated with voltage switching.

Cyclic DFG: 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 $e \in E$, 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.

Static Schedules: 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.

Retiming: Retiming [3] 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. Given a cyclic DFG G=⟨V, E_SET, d, N⟩, retiming r of G is a function from V to integers. For a node $u \in V$, the value of r(u) is the number of delays drawn from each of incoming edges of node u and pushed to all of the outgoing edges. Let G_r=⟨V, E_SET, d_r, N⟩ denote the retimed graph of G with retiming r, then $d_r(e) = d(e) + r(u) - r(v)$ for every edge $e(u \to v) \in E$.



```
E[0]=E[−1]=E[−2]=1;
A[1]=E[−2]*4+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]*4+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];
```

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

Rotation Scheduling: 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. In Figure 3, we shows an example to explain how to obtain a new schedule via rotation scheduling. We use the schedule generated by list scheduling in Figure 1(c) as an initial schedule. We get a set of nodes at the first row of the schedule that is {A}, and rotate A 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). The code size is increased by introducing the prologue and epilogue after the rotation is performed, which can be solved by the technique proposed in [7]. In the new schedule, the schedule length is reduced from 10 to 7 after the first rotation. This spare space is extremely crucial for us to relax the schedule by DVS.

4. The Scheduling Algorithms with DVS

In this section, we show our loop scheduling algorithms with DVS. The main idea of our algorithms is: Firstly, we get a shorter schedule by rotating an original schedule. Then we relax the schedule as much as possible via DVS. Here relaxing a schedule means relaxing some nodes in the schedule by reducing their voltages. The main frame of our loop scheduling algorithms is listed as follows.

```
1.  Input DFG G and environment parameter.
2.  Create a schedule ST using list scheduling.
3.  For i = 1 to R_num
4.      Get a set of nodes S to be rotated;
5.      Delete nodes in S from ST;
6.      For each $u \in S$, Retime u, and get new Gr;
7.      Compact ST according to Gr;
8.      Insert nodes in S into ST according to Gr;
9.      Relax ST using DVS;
10.     STMIN←ST if ST's energy is minimum;
11. Output STMIN.
```

Figure 4. The main frame of our algorithms.

4.1. The BALOSA Algorithm

The BAsic LOop Scheduling Algorithm (BALOSA) with DVS for low energy uses the main frame shown in Figure 5, and is described in detail as following: We first input a DFG G and the environment parameters such as the number of function-units, the timing constraint, the repeat number of rotation (R_num), the levels of voltages and the values of voltages. Then we create a schedule table ST in step 2 by using list scheduling. From step 3 to 10, there is a loop. In each loop body, we rotate the schedule and relax it. In step 4, we get a set of nodes S to be rotated, which are the nodes in the first row of ST, and remove them from ST. In step 6, we modify the retiming function r of G when we retime the nodes in S, and get a new DFG Gr. In step 7, we compact ST, which means we find an earliest position for each node in ST. Then in step 8 we put the nodes in S back to ST according to the principle of As Early As Possible (AEAP), decrease their voltages as far as the relaxations do not break either the timing constraint or the dependency relations between nodes in Gr. In step 10, we record the new schedule if its energy consumption is the minimum. After the loop of rotation and DVS, we finally get the schedule with minimum energy consumption.

4.2. The ILOSA Algorithm

The second algorithm is an Improved LOop Scheduling Algorithm (ILOSA) with greedy principles based on BALOSA. The principles designed for our second algorithm are listed as following:

The Longest Column First Principle (LCFP): This principle is used to find the node, which is at the top of the longest column of the schedule, to be rotated for the next rotation. It is a heuristic principle to find a shorter length schedule after rotation, and the shorter schedule is helpful for the further relaxation. For example, from the schedule shown in Figure 3-(b), we will choose node D to be the rotating node for the next schedule.

Clustering Relative Nodes principle (CRNP): This is a principle used to choose the nodes to be rotated together. According to this principle, if we choose a node to be rotated, we will also choose its relative nodes, which are rotatable. Two nodes are relative if they have a common post-node in DFG. This principle will make a new created schedule more compact in some cases. For example, from the schedule shown in Figure 3(b), we will rotate D, C and B for the next schedule, because they are relative and rotatable.

Larger Room for Larger Node Principle (LRLNP): This is a greedy principle designed to find suitable locations for rotated nodes. After rotation, we will consider the larger rotated node first to be inserted back into the schedule if there are more than one rotated nodes, and find the larger room for each node within timing constraint. Here a larger node means it has a larger computation cycles. We insert a node into a larger room, and it will do good for its relaxation, while an earlier position might not be a better one to its relaxation. We find the new schedule created by LRLNP will be relaxed more effectively than the one by BALOSA in many cases.

Larger Node Relaxed First Principle (LNRFP): This is a greedy principle for relaxing the schedule after we get a new schedule. Note that energy Eu is computed by $N_u C_u V_{dd}^2$. (See formula 2). It is obvious that reducing the voltage of a node with larger N_u (the number of computation cycles) can decrease more energy than the one with smaller N_u. However, we found that simply considering the size of each node's N_u is not appropriate. So we use *weighted size* to decide which node is preferable to relax. The weighted size of node u is referred to its prospective benefit size (BSu) for relaxation, and it is defined as following:

$$BSu = Nu * BFu \tag{4}$$

Here Nu is computation cycles of node u, BFu is a benefit factor of node u, and is calculated by the procedure shown in Figure 9. Since the relaxation of a node with multiple post-nodes (or pre-nodes) will affect the relaxation of its post-nodes (or pre-nodes), we will reduce its BSu by BFu, which might be less than 1.

Now we explain our second algorithm ILOSA in details. ILOSA uses the main frame shown in Figure 5, but the implementation is changed in step 4, 8 and step 9 comparing with BALOSA. In step 4 of ILOSA, to get a set of nodes to be rotated, firstly, the schedule ST is shrank and compacted, that means we assume that all the nodes are assigned the highest voltage, and put them to the earliest positions. Secondly, we get a node u from the top of the longest column of ST (if the top of the longest column is empty, we get the one from the next longer column), and add it into set S. After that, we check if u has relative nodes rotatable, and we will add them in set S if there are.

```
1. Start_Position ← The start position of node u;
2. End_Position ← The end position of node u;
3. OCell_num ← The number of cells occupied by
   some nodes in ST between Start_Position and
   End_Position;
4. TCell_num ← The total cells in ST between
   Start_Position and End_Position;
5. BFu = OCell_num / TCell_num
```

Figure 5. The implementation of calculating the benefit factor of node u.

To insert the nodes back into the schedule in step 8 of ILOSA, we put them in a queue by the descending order of their computation cycles. Then for each node in the front of the queue, we find a larger space in current ST for it with its voltage as lower as possible, and insert it into ST.

To show how to relax a schedule in step 9 of ILOSA, we list the implementation in Figure 12. First, we check the length of ST to see the timing constraint is kept or not. If the timing constraint is broken, we give up the relaxation for this ST. Otherwise; we will relax the nodes in ST as much as possible by the greedy principle LNRFP. In step 2 of Figure 12, we calculate the benefit sizes for all nodes. In step 3, we create a list named Bene_List, which is ordered by the benefit sizes of nodes descendingly, it will be used in greedy algorithm in steps 5 to 13. We also create another list: Pos_Queue, which will be used to pull back or compact ST. Here pull back ST means to put the nodes in ST back, as late as possible, to the timing limitation, as long as we keep the dependency relations between nodes in Gr for ST. From step 5 to 13, there is a repetition, where we use the greedy principle, LNRFP, to relax the nodes as much as possible within the allowance of timing constraint. In step 10, we pull back all the nodes, which are behind u in list Pos_List, to empty out the spare space for u to relax. Then, node u is relocated and relaxed in step 12, here we relocate the node u within the emptied out area. In step 13, we compact the schedule again for the rest of nodes.

5. Experiments

We have performed the experiments under following environment: a DELL PC with a P4 2.1 G processor running

```
1.  If the length of ST > Tlimit, then return;
2.  For each u ∈ V, calculate the value of BSu;
3.  Create a descending list of all nodes in V: Bene_List
    ordered by the value of benefit size of nodes;
4.  Create an ascending list of all nodes in V: Pos_List
    ordered by the start position of nodes in ST;
5.  For i = 1 to |V|
6.    u = Bene_List[i]
7.    For j = |V| to 1
8.        v = Pos_List[j]
9.        If v != u and v is behind u in Pos_List
10.           Pull back node v;
11.       Else if v == u then
12.           Relocate and Relax node u;
13. Compact ST according to Gr;
```

Figure 6. The relaxation in ILOSA.

TC	ILP([6]) Energy (μJ)	BALOSA Energy (μJ)	BALOSA %B (%)	ILOSA Energy (μJ)	ILOSA %I (%)
2-Cascaded Biquad Filter, 8 processor cores					
40	620.24	626.32	-0.98	527.44	14.96
45	551.92	476.40	13.68	452.72	17.97
50	903.60	815.60	9.74	780.88	13.58
60	642.48	680.40	-5.90	641.52	0.15
70	487.92	527.60	8.13	476.72	2.30
4-stage Lattice Filter, 8 processor cores					
50	1805.32	1023.72	40.67	877.80	51.38
70	908.20	596.52	34.32	565.32	37.75
90	2037.16	1960.68	3.75	1960.68	3.75
110	1358.12	1205.80	11.22	1205.80	11.22
120	1178.60	1960.68	3.75	1960.68	3.75
Average Imp. over ILP			11.84	–	15.68

Table 1. The energy comparison for the schedules generated by the DVS technique [6], and our BALOSA and ILOSA algorithms.

Red Hat Linux 9.0. The benchmarks include 2-Cascaded Biquad Filter, 4-stage Lattice Filter, and 8-stage Lattice Filter. The following parameters are used in the experiments: There are four discrete voltage levels: 0.9, 1.7, 2.5 and 3.3; each application has timing constraints; $C_u = 1$ in Equation 3 when the energy is calculated (this assumption is good enough to serve our purpose to compare the relative improvement among different algorithms). For each benchmarks, we conduct two sets of the experiments based on 3 processor cores and 5 processor cores, respectively. We compare our algorithms with the DVS technique in [6] that can give near-optimal solution for the DAG optimization.

The experimental results are shown in Table 1. In the table, columns "ILP[6]", "BALOSA" and "ILOSA" represent the results obtained by the DVS technique in [6], our BALOSA algorithm, and our ILOSA algorithm, respectively. Columns "%B" and "%I" represent the improvement of BALOSA and ILOSA over the DVS technique in [6]. The total average improvement of BALOSA and ILOSA is shown in the last row.

From the experimental results, we can see that our algorithms achieve more energy saving compared with the DVS technique in [6]. On average, BALOSA shows a 11.84% reduction and ILOSA shows a 15.68% reduction.

6. Conclusion

In this paper, we proposed a novel real-time loop scheduling technique to minimize the energy via loop scheduling and DVS for applications with loops. The basic idea is to repeatedly regroup a loop based on rotation scheduling and decrease the energy by DVS as much as possible within a timing constraint. We proposed two algorithms, BALOSA and ILOSA. The experimental results show that our algorithms obtain better results on energy saving than the previous DVS techniques that only consider the DAG part of a loop.

References

[1] L.-F. Chao, A. S. LaPaugh, and E. H.-M. Sha. Rotation scheduling: A loop pipelining algorithm. *IEEE Trans. On Computer-Aided Design*, 16(3):229–239, March 1997.

[2] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processor. In *ISLPED*, pages 197 –202, 1998.

[3] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:5–35, 1991.

[4] 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.

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

[6] Y. Zhang, X. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *DAC*, pages 183–188, 2002.

[7] Q. Zhuge, B. Xiao, and E. H.-M. Sha. Code size reduction technique and implementation for software-pipelined dsp applications. *ACM Transactions on Embedded Computing Systems*, 2(4):1–24, Nov. 2003.