

Energy minimization with loop fusion and multi-functional-unit scheduling for multidimensional DSP

Meikang Qiu^{a,*}, Edwin H.-M. Sha^d, Meilin Liu^e, Man Lin^b, Shaoxiong Hua^c, Laurence T. Yang^b

^aDepartment of Electrical Engineering, University of New Orleans, 2000 Lakeshore Drive, New Orleans, LA 70148, USA

^bDepartment of Computer Science, St. Francis Xavier University, Antigonish, NS, B2G 2W5, Canada

^cSynopsys, Inc., 700 East Middlefield Road, Mountain View, CA 94043, USA

^dDepartment of Computer Science, University of Texas at Dallas, Richardson, TX 75083, USA

^eDepartment of Computer Science & Engineering, Wright State University, 3640 Colonel Glenn Highway, Dayton, OH 45435, USA

Received 28 June 2006; received in revised form 14 January 2007; accepted 5 June 2007

Available online 7 November 2007

Abstract

Energy saving is becoming one of the major design issues in processor architectures with multiple *functional units* (FUs). Nested loops are usually the most critical part in multimedia and high-performance DSP systems. There is a tradeoff between power saving and performance, such as timing constraint and code size requirement, of nested loops. This paper studies how to minimize the total energy while satisfying performance requirement for applications with multidimensional nested loops. An algorithm, *energy minimization with loop fusion and FU schedule* (EMLFS), is proposed. We first use retiming and partition to fuse nested loops. Then we use novel FU scheduling algorithms to maximize energy saving without sacrificing performance. The experimental results show that the average improvement on energy saving is significant by using our EMLFS algorithm.

© 2007 Elsevier Inc. All rights reserved.

Keywords: Low-power design; Loop fusion; DSP; Schedule; Partition

1. Introduction

Embedded systems have stringent energy and area constraints due to their particular characteristics and specific usage, such as mobile computing [5,24,25,54,58]. With the advent of parallel architectures and systems with deep memory hierarchies, nested loops optimization has become an important area in high-level optimizations. In many DSP applications such as multidimensional signal processing and video applications, nested loops are the most critical sections because most of the execution time and power is spent there [25,49,59]. Loop fusion, which combines two loops into one loop, is one of the most important techniques in optimizing performance of nested loops [24,25,29,58]. Loop fusion not only improves the performance of systems, but also reduces the energy consumption of memory because it improves data reuse

and eliminates a number of memory references and branch instructions.

This paper focuses on reducing the total energy while satisfying performance constraints for loop applications. We choose to target loop optimizations for two reasons [25]: First, the multimedia and signal processing applications operate on multidimensional array structures that benefit from such optimizations. Second, these optimizations are widely used by commercial and academic optimizing compilers.

Loop fusion has two main effects on a program's demand for processor and memory resources [58]. The first effect is to reduce demand, by reducing loop overhead and by increasing data reuse in registers and cache, which in turn reduces the number of memory operations and address calculations. The second effect is to balance demand, by combining loops with different instruction mixes, cache miss rates, branch misprediction rates, etc. Reduced demand naturally tends to save both time and energy. Increased balance also tends to save time and, to a lesser extent, energy, by increasing *instruction-level parallelism* (ILP): even with aggressive clock gating in

* Corresponding author. Fax: +1 504 280 3950.

E-mail addresses: mqiu@uno.edu (M. Qiu), edsha@utdallas.edu (E.H.-M. Sha), meilin.liu@wright.edu (M. Liu), mllin@stfx.ca (M. Lin), huas@synopsys.com (S. Hua), lyang@stfx.ca (L.T. Yang).

inactive functional units, packing an operation into an unused slot in a superscalar schedule tends to save energy compared to extending program run time in order to execute the operation later.

Loop fusion can also increase system balance and has a special benefit for processors with *dynamic voltage scaling* (DVS) [58]. DVS allows CPU frequency and voltage to change dynamically at run time in order to match demand. The Transmeta Crusoe TM5800 processor can scale its frequency from 800 down to 367 MHz and its voltage from 1.3 down to 0.9 V, thereby reducing power [13]. DVS is also employed on Intel XScale processors [10]. Because required voltage scales roughly linearly with frequency within typical operating ranges, and energy is proportional to the square of the voltage, a modest reduction in supply voltage can yield significant energy saving.

For rate-based, i.e., soft real-time, applications, and for applications that are I/O or memory bound, DVS allows the processor to slow down to match the speed of the real-world or external bottleneck. Recently, researchers have proposed globally asynchronous, locally synchronous DVS processors in which the frequency and voltage of various processor components (“domains”) can be changed independently at run time [23,39,40]. Such *multiple clock domain* (MCD) processors allow domains that are not on the processor’s critical path to slow down to match the speed of an internal bottleneck. On DVS processors, loop fusion can save energy even when it does not reduce demand or improve performance, because slowing the processor down *evenly* over a long period of time will save more energy than slowing it down a lot at some times and only a little at other times [58].

In DSP applications, more and more heterogeneous FUs are embedded into a single chip. Hence, same task can be processed under heterogeneous FUs with different energy consumptions [20,42,3,12,46,57]. Therefore, an important problem arises: how to assign a FU to each task of an application such that the performance requirements can be met and the total energy consumption can be minimized.

Combining the consideration of energy and performance, in the paper, we design an algorithm, *energy minimization with loop fusion and FU schedule* (EMLFS), to minimize total energy while satisfying timing constraint for loop applications. The experimental results show that EMLFS achieves a significant reduction on average in total energy consumption. For example, with 3 FUs, compared with the list scheduling without loop fusion, EMLFS shows an average 33.5% reduction in total energy consumption.

In summary, our paper has three major contributions: First, we use novel algorithms of loop fusion to improve the energy-saving of DSP systems with the consideration of performance. Second, we exploit the multi-FU of DSP to improve energy saving. Third, we integrate these two methods into EMLFS algorithm to provide an overall energy optimization without sacrificing performance.

In the next section, we introduce necessary background. The algorithm is discussed in Section 3. We demonstrate our experimental results in Section 4. Related work and concluding remarks are provided in Sections 5 and 6, respectively.

2. Basic concepts and models

In this section, we introduce some basic concepts which will be used in the later sections. First, we give the energy model and explain why loop fusion can save energy in a view from DVS aspect. Next, we introduce our loop fusion model. Then we introduce multidimensional loop depending graph (MLDG) model and multidimensional retiming. Finally, we describe the FUs scheduling problem with an example.

2.1. The energy model and DVS

In a CMOS circuit, the dynamic power consumption is proportional to $V^2 \cdot f \cdot C_L$, where C_L is the load capacitance, V is the supply voltage, and f is the system clock frequency [45,56,43,38,37,6]. Therefore, energy is equal to $V^2 \cdot f \cdot c \cdot t$, where c is a constant and different components may have different c . Reducing the supply voltage can result in substantial power and energy saving. Roughly speaking, system’s power dissipation is halved if we reduce V by 30% without changing any other system parameters [56,16].

According to the α formula in CMOS circuit, the cycle period time T_c is proportional to $\frac{V}{(V-V_{th})^\alpha}$, where V_{th} is the threshold voltage and $\alpha \in (1.0, 2.0)$ is a technology-dependent constant [56,43,38,37]. Given the number of cycles N of node v , then its computation time $T(v) = N \times T_c$. We can see that the lower voltage will prolong the execution time of a node but reduce its energy consumption.

DVS is a technique that varies system’s operating voltages and clock frequencies based on the computation load to provide desired performance with the minimum energy consumption. It has been demonstrated as one of the most effective low-power system design techniques and has been supported by many modern microprocessors. Examples include Transmeta’s Crusoe, AMD’s K-6, Intel’s XScale and Pentium III and IV, and some DSPs developed in Bell Labs [17].

We do not consider any overhead incurred by dynamic scaling [58]. Such overhead would not change our results: because demand is spread uniformly over time in the optimal case, overhead would be zero. Here we give the basic idea of *program balance* [58]. Assuming that time is held constant, we argue that a program execution consumes minimal energy if it has constant balance. Put another way, a program with varied balance can be made more energy efficient by smoothing out the variations. Since we are assuming that components are independent, we simply need to prove this statement for a single component. For example, an execution comprising M intervals uses time $T = Mt$. The energy consumed is $E_1 = ctf \sum_{i=1}^M (V_i)^2$. The balanced execution has the same running time T and it use a constant V . It consumed energy $E_2 = cTf(V/M)^2 = ctfM(V/M)^2$. It’s easy to prove by induction that the following inequality holds: $E_1 = ctf \sum_{i=1}^M (V_i)^2 \geq ctfMV^2 = E_2$, where c , t , V_i , and V are non-negative real numbers.

2.2. Loop fusion

Loop fusion is a loop optimization technique which combines several sequential loops into one loop. We define a “ $J + K$ ” model to represent any nested loop that contains sequential K -level loops enclosed within the J -level shared outer loop as shown in Fig. 1. It is obvious that $J \geq 0$ and $K \geq 1$. When there is no shared outer loop in the “ $J + K$ ” model, $J = 0$, it becomes the “ $0 + K$ ” model. When $J = 1$, K -level sequential loops are enclosed by one shared outermost loop, and it becomes the “ $1 + K$ ” model.

Data dependency analysis is critical for loop fusion [50,31]. The fused loop should not violate any data dependency between the original loops which can be represented by the loop dependency vector. For two sequentially executed n -level nested loops A and B , with iteration $(i_1, i_2, \dots, i_n) \in B$ and $(j_1, j_2, \dots, j_n) \in A$, we say that there is a loop dependency vector $\vec{d} = (i_1 - j_1, i_2 - j_2, \dots, i_n - j_n)$ that represents the data dependency between these two loops if a data value computed in $(j_1, j_2, \dots, j_n) \in A$ is consumed in $(i_1, i_2, \dots, i_n) \in B$.

We show an example of loop fusion problem. The code shown in Fig. 2(a) contains three sequential loops enclosed in a shared outermost loop. But these loops cannot be directly fused

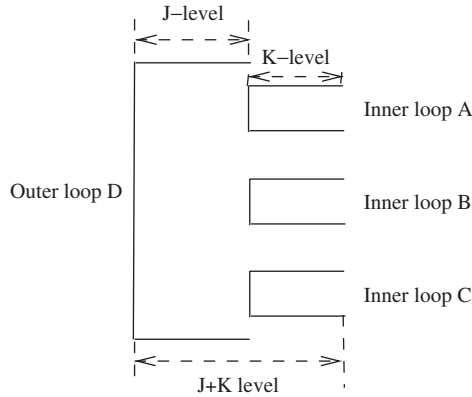


Fig. 1. The “ $J + K$ ” model.

because of the conflict data dependencies existing among them. First, we found that the loops A and B cannot be combined into one loop because we need the value of $A[i, j + 1]$ to compute $B[i, j]$. Obviously, the value $A[i, j + 1]$ is not available in the (i, j) th iteration. This kind of dependency is called fusion-prevention dependency which can be represented by a negative weight edge in a loop dependency graph (LDG) with nodes representing loops as shown in Fig. 3(a). We use the graph representation of loops in this paper and will give a detailed explanation later. Second, we found another fusion-prevention dependency between the loops A and C , i.e., the computation of $C[i, j]$ is dependent on $A[i, j + 1]$ and $A[i, j + 2]$. Thus, loops A and C cannot be fused without an appropriate transformation. Many previous works [26,32,44,35,36], cannot deal with fusion-prevention dependencies and cannot fuse loops A , B and C into one loop.

We propose a graph transformation technique to solve the loop fusion problem for the code in Fig. 2(a). By retiming nodes A , B and C in the LDG, all the negative weight edges, i.e., fusion-prevention dependencies, can be eliminated as shown in Fig. 3(b). Therefore, all the three loops can be legally fused.

2.3. Multidimensional loop dependency graph

To clearly show the data dependencies between loops, we use a LDG to model a nested loop. A MLDG $G = (V, E_d, \delta)$ is a node-weighted and edge-weighted directed graph, where V is a set of nodes representing the loops to be fused. $E_d \subseteq V \times V$ is a set of edges representing dependencies between the loops. δ is a function from E_d to Z^n representing the minimum loop dependency vector between two loops.

We use the minimum loop dependency vector of an edge in our MLDG model instead of all the loop dependency vectors to improve the efficiency of our technique. All the comparisons between two loop dependency vectors are based on the lexicographic order in this paper. For example, in the two-dimensional (2D) case, a vector $\vec{v} = (v_1, v_2)$ is smaller than a vector $\vec{u} = (u_1, u_2)$ according to the lexicographic order if either $v_1 < u_1$ or $v_1 = u_1$ and $v_2 < u_2$.

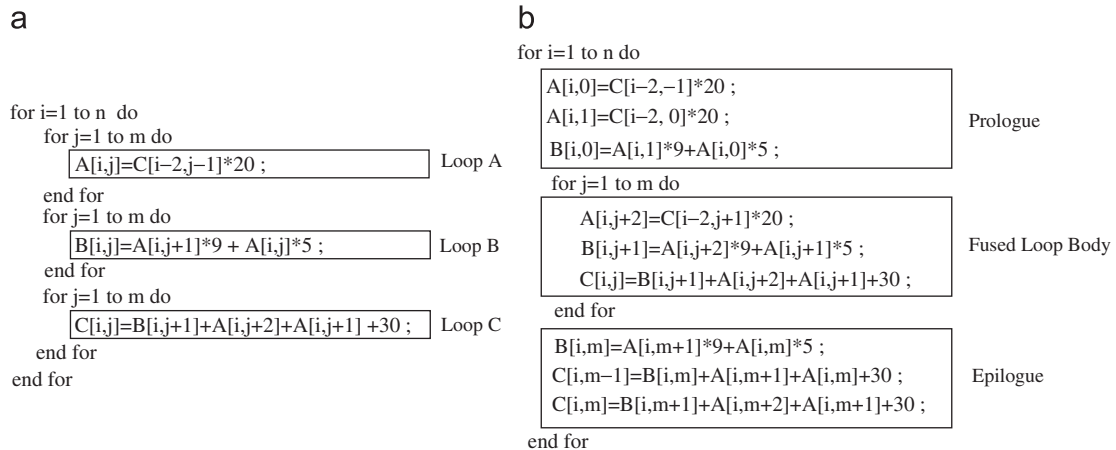


Fig. 2. (a) The original MDFG with three inner loops. (b) The fused loop.

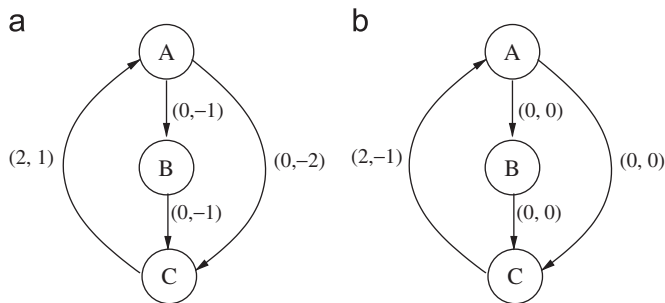


Fig. 3. (a) The LDG of the loop shown in Fig. 2(a). (b) The retimed LDG with retiming $\vec{r}(A) = (0, 2)$, $\vec{r}(B) = (0, 1)$, and $\vec{r}(C) = (0, 0)$.

In an LDG, a fusion-prevention dependency is represented by an edge e with edge weight $\delta(e) < (0, 0, \dots, 0)$. Fusion-prevention dependencies are the data dependencies which become reverse to the control flow after the loop fusion is performed, making the fusion illegal [31]. The LDG of the loop in Fig. 2(a) is shown in Fig. 3(a). There are three nodes $V = \{A, B, C\}$ in the LDG which represent the three innermost loops in the program. The loop dependency edges are $E_d = \{e_1 : A \rightarrow B, e_2 : A \rightarrow C, e_3 : B \rightarrow C, e_4 : C \rightarrow A\}$. The loop dependency vectors are $\{(0, -1), (0, 0)\}$ between nodes A and B , $\{(0, -2), (0, -1)\}$ between nodes A and C , $\{(0, 0)\}$ between nodes B and C , and $\{(2, -1)\}$ between nodes C and A . According to our MLDG definition, $\delta(e_1) = (0, -1)$, $\delta(e_2) = (0, -2)$, $\delta(e_3) = (0, 0)$, $\delta(e_4) = (2, -1)$. The fusion-prevention dependency edges are e_1 and e_2 .

2.4. Multidimensional retiming

In this paper, we develop a graph transformation technique to remove fusion-prevention dependencies based on the multidimensional retiming technique. Note that the retiming technique preserves all the data dependencies of the original LDG. Retiming redistributes delays, i.e., data dependency distances, in a graph to achieve the minimum cycle time.

For a MLDG G , a multidimensional retiming \vec{r} is a function from V to Z^n . The retiming value $\vec{r}(u)$ represents how many delays are added into the edges $u \rightarrow v$ and subtracted from the edges $w \rightarrow u$, for $u, v, w \in V$. Therefore, in the retimed MLDG G^r , we have $\delta^r(e) = \delta(e) + \vec{r}(u) - \vec{r}(v)$ for each edge $e : u \rightarrow v$. The summation of the edge weights in a cycle remains a constant after retiming. The retiming value $\vec{r}(u)$ means that a node u originally executed in the iteration \vec{i} is moved to the iteration $\vec{i} - \vec{r}(u)$. For a LDG, all the computations of loop u are executed $\vec{r}(u)$ iterations earlier. Some iterations of the original loop are moved out of the loop body to become prologue and epilogue, that is, the codes to be executed before and after the loop body to complete the execution of the whole loop. The number of copies of a node u in prologue or epilogue can be computed from the retiming value [60].

The normalized retiming value for node u is defined as $r(u) - \min_u r(u)$, where $\min_u r(u)$ is the minimum retiming value of all nodes u in V [56]. From this definition, we know that the

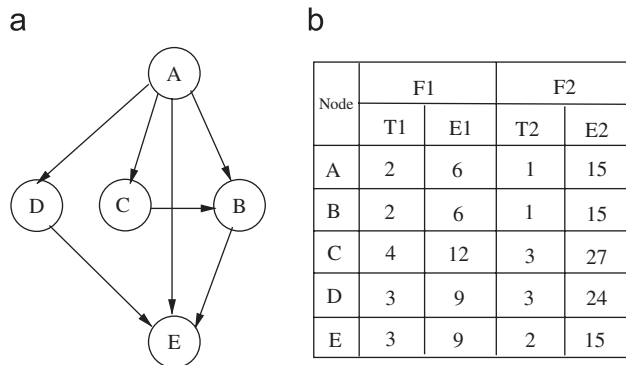


Fig. 4. (a) A DAG. (b) Execution times and energy consumption of FUs.

normalized retiming value for any node u in V is larger than or equal to $(0, 0)$ in the 2D case.

For example, the retiming values for nodes A , B , and C of the LDG in Fig. 3(a) computed by the legalizing fusion algorithm are $(0, 0)$, $(0, -1)$, and $(0, -2)$, respectively. We obtained the normalized retiming value by subtracting the minimum retiming value $(0, -2)$ from the original retiming values of the three nodes. Thus, the normalized retiming values are $\vec{r}(a) = (0, 0) - (0, -2) = (0, 2)$, $\vec{r}(b) = (0, -1) - (0, -2) = (0, 1)$, and $\vec{r}(c) = (0, -2) - (0, -2) = (0, 0)$. We retime the three nodes of the LDG in Fig. 3(a) with the normalized retiming values $\vec{r}(a) = (0, 2)$, $\vec{r}(b) = (0, 1)$, and $\vec{r}(c) = (0, 0)$. The retimed LDG G^r of the LDG in Fig. 3(a) is shown in Fig. 3(b). After retiming, the weight of edge e_1 becomes $\delta^r(e_1) = (0, -1) + (0, 2) - (0, 1) = (0, 0)$. The weight of edge e_2 becomes $\delta^r(e_2) = (0, -2) + (0, 2) - (0, 0) = (0, 0)$. And the weight of edge e_3 becomes $\delta^r(e_3) = (0, -1) + (0, 1) - (0, 0) = (0, 0)$. Therefore, all the fusion-prevention dependencies are removed.

2.5. Heterogeneous FU schedule problem

We define the heterogeneous FU scheduling problem as follows: given a heterogeneous system with M FUs, $F = F_1, \dots, F_i, \dots, F_M$, a DAG $G = \langle V, E_d \rangle$ where $V = \langle u_1, \dots, u_i, \dots, u_N \rangle$ is a set of nodes, with each node representing a task, $E_d \subseteq V \times V$ is a set of edges representing dependency relations among nodes in V , $T(u_k) = t_1(k), \dots, t_i(k), \dots, t_M(k)$, where $t_i(k)$ denotes the computation time of u_k on F_i , and a time constraint L , find a task schedule for G such that the energy consumption is minimized within L .

An example is shown in Fig. 4 and 5. Assume there is a heterogeneous system that consists of 2 heterogeneous FUs, F1 and F2. An exemplary DAG is shown in Fig. 4(a). The given time constraint for the DAG to be executed is 10 time units. The execution time and energy consumption of each node for different FUs are shown in Fig. 4(b). In Fig. 4(b), T_i denotes the execution time and E_i denotes the first part of the energy consumption E_{ij} , which is the energy consumption for u_i to be scheduled on F_j . Two schedules for the DAG are shown in Fig. 5. In Schedule 1, the schedule length is 10 time units and

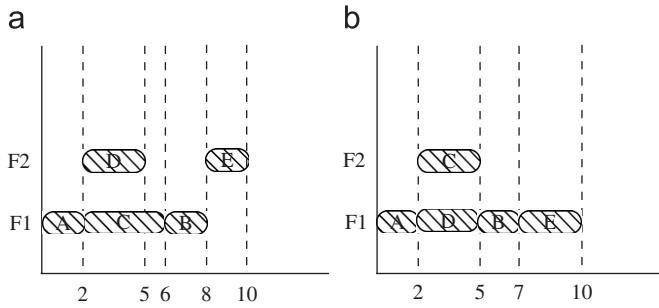


Fig. 5. (a) Schedule 1. (b) Schedule 2.

its system energy consumption is 63. In Schedule 2, the schedule length is 10 time units and its system energy consumption is 57. Both schedules satisfy the time constraint while the latter has a lower energy consumption. This example shows that different task schedules will produce different system energy consumptions.

3. The algorithms

In this section, an algorithm, EMLFS, is designed to solve the minimum total energy without sacrificing performance problem based on loop fusion and multi-FU scheduling.

3.1. The EMLFS algorithm

Algorithm 1. EMLFS.

Require: MLDG $G = \langle V, E_d, \delta \rangle$

Ensure: A schedule S and the retiming \vec{r} , to $\text{MIN}(E)$.

- 1: Use minimal partition algorithm MINP to find the minimal number of partitions.
 - 2: Use loop fusion algorithm LoopF to retime and fuse the nested loops.
 - 3: Record the retime \vec{r} .
 - 4: Based on the obtained loop function, we do multi-FU scheduling by using FU_Sch1 or FU_Sch2.
 - 5: Record the schedule S .
 - 6: Record E_{\min} .
 - 7: Output S , \vec{r} , and E_{\min} .
-

The EMLFS algorithm is shown in Algorithm 1. First, use minimal partition algorithm MINP to find the minimal number of partitions. Then use loop fusion algorithm LoopF to retime and fuse the nested loops. Next, based on the obtained loop function, we do multi-FU scheduling by using FU_Sch1 or FU_Sch2. Finally, output the schedule S , retime \vec{r} , and the minimal energy consumption E_{\min} . EMLFS algorithm has several advantages: (1) Use novel minimal partition algorithm to maximal the loop fusion. (2) Use loop fusion to exploit the timing and energy-saving improvement of a multidimensional DSP application. (3) Exploit the multi-FU structure with novel scheduling algorithms to save energy while satisfying timing constraint.

3.2. The loop fusion algorithm

Algorithm 2. LoopF algorithm.

Require: MLDG $G = \langle V, E_d, \delta \rangle$ of “1 + K ” nested loops.

Ensure: A retiming \vec{r} of the MLDG.

- 1: Remove all the edges e with $\delta_1(e) \geq 1$ from E_d , and get $G' = \langle V, E'_d, \delta \rangle$, where $E'_d = E_d - e | \delta_1(e) \geq 1$.
 - 2: Construct the constraint graph $G_c = \langle V_c, E_{d_c}, W_c \rangle$, where $V_c \leftarrow V \cup v_0$, v_0 is a pseudo source node we added.
 - 3: Add edges from v_0 to all other nodes in the constraint graph G_c with $w = 0$.
 - 4: $(\delta(e), 2) \leftarrow$ all the elements after the second dimension in $\delta(e)$.
 - 5: All the edges with $(\delta(e), 2) \geq (0, \dots, 0)$ in G will add weight of the second element of $\delta(e)$.
 - 6: All the edges with $(\delta(e), 2) < (0, \dots, 0)$ in G will add weight of the second element of $\delta(e)$ minus 1.
 - 7: Use Bellman–Ford shortest path algorithm to compute the shortest distance from v_0 to all other nodes.
 - 8: $Sh(v_i) \leftarrow$ the shortest distance from v_0 to $v_i \in G_c$.
 - 9: Set the second element of \vec{r} to be $Sh(v_i)$, and all other elements to be 0.
 - 10: Output \vec{r} .
-

For a “ $J + K$ ” nested loop, we only need to retime the $(J + 1)$ th dimension of the edge weight of the MLDG G . Hence we can transfer “ $J + K$ ” to “1 + K ” situation. Algorithm 2 gives the way to deal with “1 + K ” situation. We retime the second dimension of the MLDG of “1 + K ” model loop such that the weight of each edge in the retimed MLDG satisfies that $\delta'(e) \geq (0, \dots, 0)$. Then we obtain a graph with no fusion-prevention dependency. Thus the transformed loop is legal to be fused.

An example of a “1 + 2” model loop shown in Fig. 6(a) is used to show the graph transformation process of LoopF algorithm. Fig. 6(b) shows the corresponding constraint graph of LoopF algorithm. The normalized retiming values computed by the LoopF algorithm are $\vec{r}(a) = (0, 2, 0)$, $\vec{r}(b) = (0, 1, 0)$, and $\vec{r}(c) = (0, 0, 0)$. In the retimed MLDG shown in Fig. 6(c), there is no fusion-prevention dependency.

3.3. The minimal partition algorithm

Minimal partition for loop fusion is to minimize the number of the fusion partitions; thus the resultant number of the fused loop is minimized. Our minimal partition technique is based on the LDG and it is mapped to the graph partitioning technique. To apply minimal partition, we partition the loop nodes in the LDG into several partitions so that loop nodes connected by a fusion-preventing dependence edge are partitioned into different partitions. Thus, we guarantee that all the loop nodes inside one partition can be directly fused. Also, the number of the partition is minimized. We propose an efficient algorithm MINP which is shown in Algorithm 3.

We construct a function $Q : V \rightarrow N$, where V is the loop node set and N is a positive integer set. Q must satisfy two

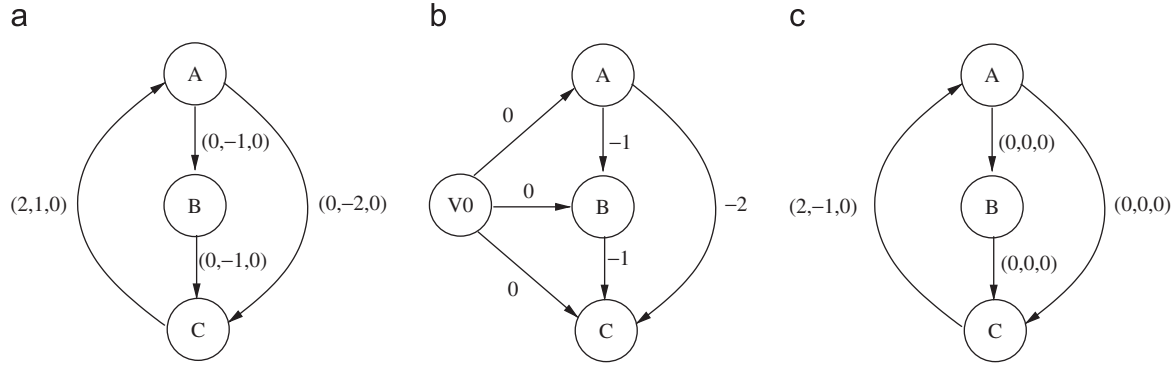


Fig. 6. (a) An example MLDG of “1 + 2” model loop. (b) The constraint graph of LoopF algorithm. (c) The retimed MLDG by the LoopF algorithm.

conditions: (i) precedence dependency constraint: for precedence dependency edge $e : u \rightarrow v, u, v \in V, Q(v) \geq Q(u)$. (ii) fusion-preventing dependence constraint: for fusion-preventing dependency edge $e : u \rightarrow v, u, v \in V, Q(v) = Q(u) + 1$.

We use shortest path algorithm to compute the partition priority of each node. The idea of our partition algorithm is as follows: First, construct a constraint graph G_c based on two kinds of constraint edges. Then apply Bellman–Ford shortest path algorithm to compute the shortest distance from v_0 to v_i in constraint graph. The nodes with the same Q were put into the same partition. Algorithm MINP gives the minimal partition number for loop fusion.

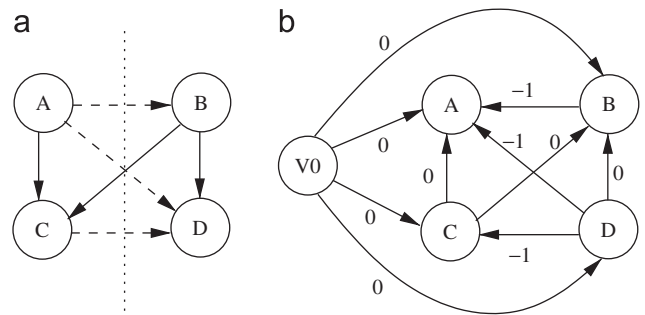


Fig. 7. (a) The LDG of a loop. (b) The constraint graph for minimal partition.

Algorithm 3. Algorithm to compute minimal partition (MINP).

Require: An LDG $G = \langle V, E_d, \delta \rangle$.

Ensure: Minimal partition of loop nodes for fusion.

- 1: $e_f \leftarrow$ the fusion-preventing edge.
- 2: $e_d \leftarrow$ the precedence dependency edge.
- 3: $G_c \leftarrow$ the constraint graph.
- 4: Remove all the edges e with $\delta_1(e) \geq 1$ from E_d , and get $G' = \langle V, E'_d, \delta \rangle$, where $E'_d = E_d - e | \delta_1(e) \geq 1$.
- 5: Add one corresponding node Q_i in the G_c for each node v_i in G' .
- 6: **for all** $e' \in G'$, where e' is from node $i \rightarrow$ to node j , **do**
- 7: **if** $e' \in e_f$ **then**
- 8: Add one edge $Q_j \rightarrow Q_i$ in G_c with weight $w = -1$.
- 9: **else**
- 10: Add one edge $Q_j \rightarrow Q_i$ in G_c with $w = 0$.
- 11: **end if**
- 12: **end for**
- 13: /* Use Bellman–Ford shortest path algorithm */
- 14: Add a pseudo source node v_0 , and the edges from it to all other nodes in the constraint graph G_c with $w = 0$.
- 15: $Sh(v_i) \leftarrow$ the shortest distance from v_0 to $v_i \in G_c$.
- 16: $Sh(v_i) \leftarrow -k_i$.
- 17: $K_{\max} \leftarrow \max(k_i)$.
- 18: There are $K_{\max} + 1$ partitions. Put the nodes with same Q into the same partition.

In Algorithm 3, we have two kinds of edges: fusion-prevention edges e_f and dependency edges. e_d in G . After the construction of constraint graph and the shortest path algorithm on the constraint graph, let $Sh(v_i)$ be the shortest distance between source node v_0 to node v_i , where v_0 is a

pseudo node. Note that there must exist a node v such that $Sh(v) = 0$, because G is a DAG. Hence, the constraint graph must be a DAG, too. Let $Sh(v_i) = -k_i$ where $k_i \geq 0$ and let K_{\max} be the maximum of all k_i . We know that the edge is either e_f or e_d , and then the weights between them are either -1 or 0 in the new constraint graph. Thus, the values of $Sh()$ must be continuous integers. Thus, there are $K_{\max} + 1$ different values among all k_i . It is obvious that the number of partitions from our algorithm is $K_{\max} + 1$. Here, we want to prove it is impossible to have less than $K_{\max} + 1$ partitions as a legal output of partitions, i.e., $K_{\max} + 1$ is the lower bound. Consider node v_m whose $Sh(v_m) = -K_{\max}$. From the definition of the shortest path, there must exist a path with K_{\max} edges of e_f (weight = -1) from v_0 to v_m . Along this path, it is obvious that all the nodes associated with these K_{\max} edges of e_f must be put into $K_{\max} + 1$ partitions. This path from the shortest path algorithm has given us the lower bound of the number of partitions between v_0 to v_m . So it is impossible to have less than $K_{\max} + 1$ partitions by simply considering this path from v_0 to v_m . Therefore, we can get the conclusion that the partition number of each loop node computed by partition algorithm is the minimal.

For example, for the LDG shown in Fig. 7(a), the constructed constraint graph is shown in Fig. 7(b). After implementing our minimal partition algorithm, we get two partition groups: $partition_1 = \{B, D\}$ with $Q = 0$ and $partition_2 = \{A, C\}$ with $Q = 1$.

Table 1
Symbols in the FU scheduling algorithms

Symbol	Meaning
Min	Minimum energy consumption for the current node
EST_i	Earliest starting time for node i
LST_i	Latest starting time for node i
FT_i	Finish time of node i
SL_j	Schedule length for FU_j
X_i	The FU that node i is scheduled on
E_{total}	Overall energy consumption of the system

3.4. The FU schedule algorithm

Scheduling problems with time and resource constraints are well known to be NP complete. We are going to solve a scheduling problem with time and resource constraints in a heterogeneous system and minimize the energy of the system at the same time. Therefore, our problem is also NP complete. In this section, two heuristic algorithms, FU_Sch1 and FU_Sch2, are developed to solve this problem. FU_Sch1 uses a bipartite matching strategy based on ALAP scheduling and FU_Sch2 uses a progressive relaxation strategy based on ALAP scheduling. Some symbols used in our algorithms are listed in Table 1.

3.4.1. FU scheduling with bipartite matching

FU_Sch1 is designed to use the bipartite matching strategy to schedule tasks. The idea is: first, use the ALAP scheduling, which minimizes the schedule length, to get the latest starting time for every node. Then construct a bipartite matching graph with the nodes in the ready list in one set and all FUs in the other set. Then reschedule nodes based on the minimum energy bipartite matching. This algorithm is shown in Algorithm 4. V_1 and V_2 in the algorithm represent the two sets used in the bipartite matching.

This algorithm first schedules all nodes using the ALAP scheduling. Based on the schedule, we use the bipartite matching to schedule nodes. For each FU, among those nodes not marked by matching, the node with the earliest start time is considered: if it has no dependency constraint at that time, it is inserted into the ready list. A bipartite matching graph is constructed as follows: all nodes from the ready list are on one side, denoted by a set V_1 , and all FUs are on the other side, denoted by a set V_2 . Each node, u_i , in V_1 has an edge connected with each FU, F_j , in V_2 . If the schedule length of the FU plus the node's computation time is less than the finish time of the node in ALAP, the edge weight is set to the energy E_{ij} . Otherwise, the edge weight is set to be infinity. After constructing the graph, call the minimum-cost-bipartite-matching function to get a minimum cost bipartite matching. Since the edge weight is set to the energy, the matching produced by the function minimizes the energy in each scheduling step. After a match is found, schedule the tasks on the corresponding FUs, mark the node, and update their descendants' information, i.e., dependence constraints and earliest starting time, recursively. Repeat this process until there is no more node to be

rescheduled. Because the FU selection for a node is limited by the finish time in the ALAP, the node can at least be scheduled on the same FU as ALAP. Thus, as long as there is an ALAP schedule for the graph, this algorithm will not fail to produce a schedule for all the tasks. As the energy is the matching factor in the bipartite matching, the energy is improved over the list scheduling algorithm.

Algorithm 4. FU_Sch1 algorithm.

Require: A DAG $G = \langle V, E_d \rangle$, a set of FUs, a timing constraint L .

Ensure: A task scheduling with energy saving.

```

1: for all  $u_i \in V$  do
2:    $EST_i \leftarrow$  starting time of node  $i$  in ASAP;
3:    $LST_i \leftarrow$  starting time of node  $i$  in ALAP;
4:    $FT_i \leftarrow$  finish time of node  $i$  in ALAP;
5: end for
6: for all  $F_j \in F$  do
7:    $SL_j \leftarrow 0$ ;
8: end for
9:  $E_{total} \leftarrow 0$ ;
10: while  $\exists$  nodes not marked do
11:   for all  $F_j \in F$  do
12:     if the first node in  $F_j$  has no dependency constraint then
13:       put it into  $V_1$ ;
14:     end if
15:     put  $F_j$  into  $V_2$ ;
16:   end for
17:   construct a weighted bipartite matching graph  $G_{BM} = \langle V_{BM}, E_{BM} \rangle$ ;
18:    $V_{BM} = V_1 \cup V_2$ ;
19:   for all  $u_i \in V_1$  do
20:     for all  $F_j \in V_2$  do
21:       compute  $E_{ij}$ ;
22:       add an edge  $e_{ij}$  between  $u_i$  and  $F_j$  into  $E_{BM}$ ;
23:       if  $SL_j + t_j(i) +$  data migration delay  $\leq FT_i$  then
24:         set  $E_{ij}$  as the edge weight;
25:       else
26:         set the edge weight as infinity;
27:       end if
28:     end for
29:   end for
30:    $M \leftarrow$  minimum cost bipartite matching for nodes in  $G_{BM}$ ;
31:   for all  $e_{ij}$  in the matching  $M$  do
32:     mark  $u_{ij}$  as scheduled;  $X_i \leftarrow$  the matching  $F_j$ ;
33:      $E_{total} \leftarrow E_{ij} + E_{total}$ ;
34:      $SL_{X_i} \leftarrow \max(EST_i, SL_j) + t_{X_i}(i)$ ;
35:     add the data migration delay into  $SL_{X_i}$ ;
36:     for all  $u_k$  which is a dependent of  $u_i$  do
37:       update the dependence information for  $u_k$ ;
38:       if  $EST_k < SL_{X_i}$  then
39:          $EST_k \leftarrow SL_{X_i}$ ;
40:       end if
41:     end for
42:   end while

```

3.4.2. FU scheduling with progressive relaxation

FU_Sch2 progressively improves the energy saving based on the schedule obtained by the ALAP scheduling. The idea is to reschedule each node to reduce the system energy consumption as much as possible. It is shown in Algorithm 5. After the initialization, this algorithm first obtains an ALAP schedule for all the tasks in the graph. Then the following steps are repeated until all nodes are marked: among all nodes that are not

Algorithm 5. FU_Sch2 algorithm.**Require:** a DAG $G = \langle V, E_d \rangle$, a set of FUs, a timing constraint L .**Ensure:** A task scheduling with energy saving.

```

1: for all  $u_i \in V$  do
2:    $X_i \leftarrow -1$ ;
3:    $EST_i \leftarrow$  starting time of node  $i$  in ASAP;
4:    $LST_i \leftarrow$  starting time of node  $i$  in ALAP;
5:    $FT_i \leftarrow$  finish time of node  $i$  in ALAP;
6: end for
7: for all  $F_j \in F$  do
8:    $SL_j \leftarrow 0$ ;
9: end for
10:  $E_{total} \leftarrow 0$ ;
11: while  $\exists$  nodes not marked do
12:    $Min \leftarrow \infty$ ;
13:   take the node  $u_i$ , with minimum  $LST_i$  and not marked;
14:   for all  $F_j \in F$  do
15:     compute  $E_{ij}$ ;
16:     if  $SL_j + t_j(i) +$  data migration delay  $\leq FT_i$  then
17:       if  $E_{ij} < Min$  then
18:          $Min \leftarrow E_{ij}$ ;  $X_i \leftarrow F_j$ ;
19:       end if
20:       if  $E_{ij} = Min$  then
21:          $X_i \leftarrow$  the FU with earlier finish time;
22:       end if
23:     end if
24:   end for
25:   mark  $u_{ij}$  as scheduled;
26:    $E_{total} \leftarrow Min + E_{total}$ ;
27:    $SL_{X_i} \leftarrow \max(EST_i, SL_j) + t_{X_i}(i)$ ;
28:   add the data migration delay into  $SL_{X_i}$ ;
29:   for all  $u_k$  which is a dependent of  $u_i$  do
30:     update the dependence information for  $u_k$ ;
31:     if  $EST_k < SL_{X_i}$  then
32:        $EST_k \leftarrow SL_{X_i}$ ;
33:     end if
34:   end for
35: end while

```

marked, take the node with earliest starting time and reschedule it to a FU such that the system energy consumption is minimized. A node can only be rescheduled to a FU if its finish time is earlier than that in ALAP and the task does not overlap with other tasks remaining in the ALAP schedule. The choice of the node is made this way because the remaining nodes can always be scheduled within the time constraint as long as the ALAP schedule exists for this task graph. After this task is scheduled, mark the node, update the dependence constraint information and the earliest start time for all its descendants recursively, update the system energy consumption and the schedule length, then continue this process until all nodes are marked.

4. Experiments

In this section, we conduct experiments with the EMLFS algorithm on a set of benchmarks including wave digital filter (WDF), infinite impulse filter (IIR), differential pulse-code modulation (DPCM) device, 2D filter, Floyd–Steinberg algorithm (Floyd), and all-pole filter. We build a simulation framework to evaluate the effectiveness of our approach. K different FU types, F_1, \dots, F_K , are used in the system, in

which a FU with type F_1 is the quickest with the highest energy consumption and a FU with type F_K is the slowest with the lowest energy consumption. Each task node has different energy consumptions under different FU types. Due to the page limit, we do not list the detailed table in this paper. We conducted experiments on six methods: Method 1: list scheduling, without loop fusion; Method 2: list scheduling, with loop fusion; Method 3: FU_Sch1 without loop fusion; Method 4: our EMLFS algorithm (with FU_Sch1); Method 5: FU_Sch2 without loop fusion; and Method 6: our EMLFS algorithm (with FU_Sch2). In the list scheduling, the priority of a node is set as the longest path from this node to a leaf node [33]. The experiments are performed on a Dell PC with a P4 2.1 G processor and 512 MB memory running Red Hat Linux 9.0.

The experimental results for the six methods are shown in Tables 2–4 when the number of FUs is 3, 4, and 5, respectively. Column “Bench.” stands for the benchmarks we used in the experiments. Column “N.” represents the number of nodes of each filter benchmark. Column “Med. 1” to “Med. 6” represents the six methods we used in the experiments. Column “E” represents the minimum total energy consumption obtained from six different methods. Column “% M1”, “% M2” under “Med. 6” represents the percentage of reduction in total energy consumption, compared to Methods 1 and 2, respectively. The average reduction is shown in the last row of the table.

The results show that our algorithm EMLFS can significantly improve the performance of multi-FU multidimensional DSP. All the methods based on our algorithm (Methods 4 and 6) improve the energy reduction over the traditional list scheduling algorithm. Among them, EMLFS with FU_Sch2 (Method 6) gives the best performance. We can see that with more FUs selections, the reduction ratio for the total energy consumption has increased. For example, with 3 FUs, compared with Method 1, EMLFS with FU_Sch2 (Method 6) shows an average 33.6% reduction in total energy consumption. While using 5 FUs, the reduction rate changed to be 36.1% for total energy consumption.

It is worthwhile to point out that we obtain this improvement ratio without sacrificing performance. For example, Table 5 shows the energy consumption vs. time constraints by benchmark IIR filter. EMLFS with FU_Sch2 (Method 6) improves the energy reduction significantly when the time constraint is large. If the time constraint is small, it still improves the energy reduction while meeting the constraint. EMLFS with FU_Sch1 (Method 4) can always improve energy reduction as long as there exists a schedule by the ALAP scheduling. The improvement is not as significant as EMLFS with FU_Sch2 (Method 6) when the time constraint is large. This is because the algorithm always tries to use all available FUs in each step, while EMLFS with FU_Sch2 (Method 6) schedules one node in each step and avoids FUs with high energy consumption if possible.

In conclusion, our algorithm has three main pros: First, we use novel algorithms of loop fusion to improve the energy saving of DSP systems with the consideration of performance. Second, we exploit the multi-FUs of DSP to improve energy

Table 2

The comparison of total energy consumption with six methods on various benchmarks when the time constraint is 1000

Bench.	N.	Six methods comparison with 3 FUs							
		Med. 1	Med. 2	Med. 3	Med. 4	Med. 5	Med. 6	% M1 (%)	% M2 (%)
		E (μ J)	E (μ J)	E (μ J)	E (μ J)	E (μ J)	E (μ J)		
WDF(1)	4	480	422	402	341	368	325	32.3	23.0
WDF(2)	12	1435	1238	1173	1018	1072	992	30.9	19.9
IIR	16	1906	1625	1511	1310	1405	1247	34.6	23.3
DPCM	16	1932	1657	1536	1361	1431	1276	34.0	23.0
2D(1)	34	4082	3520	3326	2901	3087	2872	29.6	18.4
2D(2)	4	518	448	421	382	403	301	41.9	32.8
MDFG1	8	1027	879	812	705	736	748	27.2	14.9
MDFG2	8	1142	976	901	798	824	724	36.6	25.8
Floyd	16	2010	1721	1621	1403	1483	1321	34.3	23.2
All-pole	29	3451	2951	2781	2431	2549	2276	34.0	22.9
Average reduction (%)								33.6	22.8

Table 3

The comparison of total energy consumption with six methods on various benchmarks when the time constraint is 1000

Bench.	N.	Six methods comparison with 4 FUs							
		Med. 1	Med. 2	Med. 3	Med. 4	Med. 5	Med. 6	% M1 (%)	% M2 (%)
		E (μ J)	E (μ J)	E (μ J)	E (μ J)	E (μ J)	E (μ J)		
WDF(1)	4	547	482	451	388	424	362	33.8	24.9
WDF(2)	12	1658	1467	1385	1175	1269	1089	34.3	25.8
IIR	16	2305	1965	1832	1642	1786	1462	36.6	25.6
DPCM	16	2387	2016	1901	1698	1762	1502	37.1	25.5
2D(1)	34	4872	4198	4025	3465	3703	3305	32.2	21.3
2D(2)	4	628	532	476	431	452	406	35.4	23.7
MDFG1	8	1259	1092	1038	891	958	832	33.9	23.8
MDFG2	8	1326	1142	1092	948	1002	843	36.4	26.2
Floyd	16	2518	2166	2041	1802	1891	1662	34.0	23.3
All-pole	29	4086	3518	3312	2926	3075	2630	35.6	25.2
Average reduction (%)								34.9	24.5

Table 4

The comparison of total energy consumption with six methods on various benchmarks when the time constraint is 1000

Bench.	N.	Six methods comparison with 5 FUs							
		Med. 1	Med. 2	Med. 3	Med. 4	Med. 5	Med. 6	% M1 (%)	% M2 (%)
		E (μ J)	E (μ J)	E (μ J)	E (μ J)	E (μ J)	E (μ J)		
WDF(1)	4	711	632	585	512	542	462	35.0	26.9
WDF(2)	12	2155	1865	1793	1543	1638	1395	35.3	25.2
IIR	16	2996	2561	2392	2101	2219	1883	37.2	26.5
DPCM	16	3125	2673	2501	2197	2312	2012	35.6	24.7
2D(1)	34	6233	5372	5107	4515	4791	3924	37.0	27.0
2D(2)	4	812	681	624	546	597	516	36.5	24.2
MDFG1	8	1711	1492	1408	1224	1285	1102	35.6	26.1
MDFG2	8	1785	1511	1411	1253	1318	1120	37.3	25.9
Floyd	16	3262	2718	2603	2278	2410	2087	36.0	23.2
All-pole	29	5276	4561	4312	3816	4011	3382	35.9	25.9
Average reduction (%)								36.1	25.6

Table 5
The comparison of total energy consumption with six methods on IIR filter under different time constraints

Time	Six methods comparison on IIR with 3 FUs							
	Med. 1	Med. 2	Med. 3	Med. 4	Med. 5	Med. 6		
	E (μJ)	E (μJ)	E (μJ)	E (μJ)	E (μJ)	E (μJ)	% M1 (%)	% M2 (%)
300	2096	2031	1964	1808	1827	1796	14.3	11.6
400	2058	1950	1889	1729	1756	1683	18.2	13.7
500	2039	1885	1813	1638	1686	1609	21.1	14.7
600	2020	1820	1738	1572	1616	1521	24.7	16.4
700	2001	1788	1677	1507	1560	1447	27.7	19.1
800	1963	1723	1617	1441	1503	1372	30.1	20.4
900	1925	1674	1556	1376	1447	1309	32.0	21.8
1000	1906	1625	1511	1310	1405	1247	34.6	23.3
1100	1887	1543	1466	1271	1363	1210	35.8	23.7
1200	1848	1462	1431	1247	1335	1185	35.9	23.8

saving. Third, we integrate these two methods into EMLFS algorithm to provide an overall energy optimization without sacrificing performance.

5. Related work

Loop fusion: Several valuable work has been proposed for loop fusion [31,47]. Early work includes that of Wolfe [51] and of Allen and Kennedy [1]. Global loop fusion was formulated as a graph problem by Gao et al. [14] for register reuse. Loop fusion has been extended and evaluated in many other studies that we do not have space to enumerate. Ding and Kennedy used reuse-based fusion to fuse loops of different control structures [11]. Loop shifting and loop alignment try to compute loop index offsets for loop fusion [51]. Unnecessary shifting may be introduced because their dependence graph model is built on a lower level abstraction. “Shift-and-peel” loop transformation [31] is proposed by Manjikian et al. The technique requires uniform dependencies among the loops to deal with fusion-prevention dependencies.

Leiserson and Saxe proposed the retiming technique [27]. Passos et al. developed the multidimensional retiming technique to optimize the multidimensional problems [34]. Sha et al. proposed a general loop fusion method [29]. Whereas these fusion studies were aimed at improving performance on conventional machines, our work is aimed at saving energy on DVS and multi-FU DSP processors. Also, our graph transformation technique combined with LDG model provides an instant solution of loop fusion. As a result, our algorithm always finds an optimal solution within just one round. Furthermore, the complexity remains constant no matter how deep the loop nests are.

Heterogeneous scheduling and assignment: Heterogeneous assignment of special purpose architectures for real-time DSP applications has become a common and critical step in the design flow in order to satisfy the requirements of high sample rates or stringent timing constraint [8,21,48,7]. DSP applications need special high-speed FUs, like adders and multipliers to perform addition and multiplication operations [42]. Energy-saving task scheduling in multi-FU DSP systems has

been mostly on homogeneous multiprocessors [56,9,2]; few results considered heterogeneous systems in energy-saving real-time task scheduling [42,53,30]. Among the work for heterogeneous multi-FU DSP systems, Yu and Prasanna [53] considered the minimization of energy consumption for systems. The proposed algorithm is based on the *integer linear programming* (ILP) without guarantees on the final solution. Luo and Jha [30] proposed heuristics based on list scheduling for the scheduling of real-time tasks in heterogeneous distributed systems. However, little existing work for energy-saving scheduling in heterogeneous multi-FU systems provides guarantees on the energy consumption.

Dynamic voltage and frequency scaling: Many researchers have studied on DVS [45,56,43,38,37,6]. Semeraro et al. designed a multiple cycle domain (MCD) under different time constraints system to support fine-grained DVS within a processor [58,39,40]. Iyer et al. proposed a similar design [23]. Yao et al. [52,28] and Ishihara et al. [19] studied the optimal schedule for DVS processors in the context of energy-efficient task scheduling. Both showed that it is most energy efficient to use the lowest frequency that allows an execution to finish before a given deadline. Ishihara et al. also showed that, when only discrete frequencies are allowed, the best schedule is to alternate between at most two frequencies.

The International Technology Roadmap for Semiconductors [22] predicts that the future system will feature multiple supply voltages (V_{dd}) and multiple threshold voltages (V_{th}) on the same chip [15]. This enables the DVS, which varies the supply voltage according to workload at run time [52,19]. The highest energy efficiency is achieved when voltage can be varied arbitrarily [4]. However, physical constraints of CMOS circuit limit the applicability of having voltage varying continuously. Instead, it is more practical to make multiple discrete voltages simultaneously available for the system [18]. There are several papers propose effective ways to reduce energy consumption with DVS with the consideration of leakage power [55,41]. Comparing with our approach, the authors in [55] do not consider heterogeneous FU scheduling and assignment. Furthermore, our approach has particular effects to the applications with nested loops. For example, Shen et al. [41] use

dynamic critical path predictor to reduce energy consumption. Their method is not quite practical to the applications with nested loops.

6. Conclusion

In this paper, we studied the scheduling and assignment problem that minimizes the total energy without sacrificing performance on multidimensional nested loops. We proposed a highly efficient algorithm, EMLFS, for applications with loops. By combining loop fusion and multi-FU scheduling, our algorithm improves both the performance and energy saving of multi-dimensional DSP applications. A wide range of benchmarks have been tested on the experiments. The experimental results showed that our algorithm significantly improved both the energy saving and performance for applications on nested loops.

Acknowledgments

This work is partially supported by TI University Program, NSF EIA-0103709, Texas ARP 009741-0028-2001, NSF CCR-0309461, NSF IIS-0513669, and Microsoft, USA.

References

- [1] J.R. Allen, K. Kennedy, Vector register allocation, *IEEE Trans. Comput.* 41 (10) (2002) 1290–1317.
- [2] H. Aydin, R. Melhem, D. Mosse, P. Alvarez, Dynamic and aggressive scheduling techniques for power aware real-time systems, in: *RTSS*, 2001.
- [3] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, Y. Robert, Scheduling strategies for master–slave tasking on heterogeneous processor platforms, *IEEE Trans. Parallel Distributed Systems* 15 (4) (2004).
- [4] T. Burd, T. Pering, A. Stratakos, R. Brodersen, A dynamic voltage scaled microprocessor system, *IEEE J. Solid-State Circuits* 35 (11) (2000).
- [5] F. Catthoor, S. Wuytack, E.D. Greef, F. Balasa, L. Nachtergaele, A. Vandecappelle, Custom Memory Management Methodology—Exploration of Memory Organization for Embedded Multimedia System Design, Kluwer Academic Publishers, Dordrecht, June 1998.
- [6] A. Chandrakasan, S. Sheng, R. Brodersen, Low-power cmos digital design, *IEEE J. Solid-State Circuits* 27 (4) (1992).
- [7] L.-F. Chao, A. LaPaugh, E.H.-M. Sha, Rotation scheduling: a loop pipelining algorithm, *IEEE Trans. Comput.-Aided Design* 16 (1997).
- [8] L.-F. Chao, E.H.-M. Sha, Static scheduling for synthesis of dsp algorithms on various models, *J. VLSI Signal Process. Systems for Signal Image and Video Tech.* 10 (1995).
- [9] J.-J. Chen, T.-W. Kuo, Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics, in: *ICPP*, 2005.
- [10] L.T. Clark, Circuit design of xscale™ microprocessors, in: *Symposium on VLSI Circuits, Short Course on Physical Design for Low-Power and High-Performance Microprocessor Circuits*, June 2001.
- [11] C. Ding, K. Kennedy, Improving effective bandwidth through compiler enhancement of global cache reuse, *J. Parallel and Distributed Comput.* 64 (1) (2004) 108–134.
- [12] A. Dogan, F. Özgüner, Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing, *IEEE T. Parall. Distr.* 13 (3) (2002) 308–323.
- [13] M. Fleischmann, Crusoe power management—reducing the operating power with longrun, in: *The 12th HOT CHIPS Symposium*, August 2000.
- [14] G. Gao, R. Olsen, V. Sarkar, R. Thekkath, Collective loop fusion for array contraction, in: *The 5th Workshop on Languages and Compilers for Parallel Computing*, August 1992.
- [15] S. Hua, G. Qu, Voltage set-up problem for embedded systems with multiple voltages, *IEEE Trans. Very Large Scale Integration (VLSI) Systems* 13 (7) (2005).
- [16] S. Hua, G. Qu, S.S. Bhattacharyya, Energy reduction techniques for multimedia applications with tolerance to deadline misses, in: *ACM/IEEE Design Automation Conference (DAC)*, 2003.
- [17] S. Hua, G. Qu, S.S. Bhattacharyya, Exploring the probabilistic design space of multimedia systems, in: *IEEE International Workshop on Rapid System Prototyping*, 2003.
- [18] Intel, The Intel XScale microarchitecture, Technical Summary, 2000.
- [19] T. Ishihara, H. Yasuura, Voltage scheduling problem for dynamically variable voltage processor, in: *ISLPED*, 1998.
- [20] K. Ito, L. Lucke, K. Parhi, Ilp-based cost-optimal dsp synthesis with module selection and data format conversion, *IEEE Trans. Very Large Scale Integration (VLSI) Systems* 6 (1998).
- [21] K. Ito, K. Parhi, Register minimization in cost-optimal synthesis of dsp architecture, in: *Proceedings of the IEEE VLSI Signal Processing Workshop*, October 1995.
- [22] ITRS, International Technology Roadmap for Semiconductors, International SEMATECH, Austin, TX (<http://public.itrs.net/>).
- [23] A. Iyer, D. Marculescu, Power-performance evaluation of globally asynchronous, locally synchronous processors, in: *The 29th International Symposium on Computer Architecture*, May 2002.
- [24] M. Kandemir, S. Son, G. Chen, An evaluation of code and data optimizations in the context of disk power reduction, in: *ISLPED*, 2005.
- [25] M. Kandemir, N. Vijaykrishnan, M. Irwin, W. Ye, Influence of compiler optimizations on system power, in: *Design Automation Conference (DAC)*, 2000.
- [26] K. Kennedy, K.S. Mckinley, Maximizing loop parallelism and improving data locality via loop fusion and distribution, in: *Languages and Compilers for Parallel Computing, Lecture Notes in Computer Science*, vol. 768, 1993.
- [27] C.E. Leiserson, J.B. Saxe, Retiming synchronous circuitry, *Algorithmica* 6 (1991).
- [28] M. Li, F. Yao, An efficient algorithm for computing optimal discrete voltage schedules, *SIAM J. Comput.* 35 (3) (2005).
- [29] M. Liu, Q. Zhuge, Z. Shao, E.H.-M. Sha, General loop fusion technique for nested loops considering timing and code size, in: *CASES*, 2004.
- [30] J. Luo, N. Jha, Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems, in: *VLSID*, 2002.
- [31] N. Manjikian, T.S. Abdelrahman, Fusion of loops for parallelism and locality, *IEEE Transactions on Parallel and Distributed Systems* 8 (1997).
- [32] N. Megiddo, V. Sarkar, Optimal weighted loop fusion for parallel programs, in: *The 9th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1997.
- [33] G.D. Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, 1994.
- [34] N. Passos, E.H.-M. Sha, Achieving full parallelism using multi-dimensional retiming, *IEEE Trans. Parallel Distributed Systems* 7 (11) (1996).
- [35] Y. Qian, S. Carr, P. Sweany, Loop fusion for clustered vliw architecture, in: *LCTES-SCOPES*, 2002.
- [36] Y. Qian, S. Carr, P. Sweany, Optimizing loop performance for clustered vliw architectures, in: *IEEE PACT*, 2002.
- [37] T. Sakurai, A.R. Newton, Alpha-power law mosfet model and its application to cmos inverter delay and other formulas, *IEEE J. Solid-State Circuits* SC-25 (2) (1990).
- [38] H. Saputra, M. Kandemir, N. Vijaykrishnan, M.J. Irwin, J.S. Hu, C.-H. Hsu, U. Kremer, Energy-conscious compilation based on voltage scaling, in: *LCTES'02*, June 2002.
- [39] G. Semeraro, D. Albonese, S. Dropsho, G. Magklis, S. Dwarkadas, M. Scott, Dynamic frequency and voltage control for a multiple clock domain microarchitecture, in: *The 35th International Symposium on Microarchitecture*, November 2002.
- [40] G. Semeraro, G. Magklis, R. Balasubramonian, D. Albonese, S. Dwarkadas, M. Scott, Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling, in: *The 8th*

International Symposium on High-Performance Computer Architecture, February 2002.

- [41] J. Seng, E. Tune, D. Tullsen, Reducing power with dynamic critical path information, in: MICRO 2001, 2001.
- [42] Z. Shao, Q. Zhuge, C. Xue, E.H.-M. Sha, Efficient assignment and scheduling for heterogeneous dsp systems, IEEE T. Paralle. Distr. 16 (2005).
- [43] D. Shin, J. Kim, S. Lee, Low-energy intra-task voltage scheduling using static timing analysis, in: Design Automation Conference (DAC), 2001.
- [44] S.K. Singhai, K.S. McKinley, A parameterized loop fusion algorithm for improving parallelism and cache locality, Comput. J. 40 (6) (1997).
- [45] M.R. Stan, W.P. Bursleson, Bus-invert coding for low-power i/o, IEEE Trans. Very Large Scale Integration (VLSI) Systems 3 (1) (1995).
- [46] S. Tongsima, E.H.-M. Sha, C. Chantrapornchai, D. Surma, N. Passos, Probabilistic loop scheduling for applications with uncertain execution time, IEEE Trans. Comput. 49 (2000).
- [47] S. Verdoolaege, M. Bruynooghe, F. Catthoor, Multi-dimensional incremental loop fusion for data locality, in: ASAP, 2003.
- [48] C.-Y. Wang, K.K. Parhi, Resource constrained loop list scheduler for dsp algorithms, J. VLSI Signal Process. 11 (1995).
- [49] Z. Wang, X. Hu, Energy-aware variable partitioning and instruction scheduling for multibank memory architectures, ACM Trans. Design Automat. of Electron. Systems (TODAES) 10 (2) (2005).
- [50] M.E. Wolfe, High Performance Compilers for Parallel Computing, Addison-Wesley, Redwood City, CA, 1996.
- [51] M.J. Wolfe, Optimizing supercompilers for supercomputers, Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, October 1982.
- [52] F. Yao, A. Demers, S. Shenker, A scheduling model for reduced cpu energy, in: The 36th Symposium on Foundations of Computer Science (FOCS), Milwaukee, WI, October 1995.
- [53] Y. Yu, V.K. Prasanna, Power-aware resource allocation for independent tasks in heterogeneous real-time systems, in: ICPADS, 2002.
- [54] J. Zambreno, M. Kandemir, A. Choudhary, Enhancing compiler techniques for memory energy optimizations, in: EMSOFT, 2002.
- [55] W. Zhang, N. Vijaykrishnan, M. Kandemir, M. Irwin, D. Duarte, Y. Tsai, Exploiting vliw schedule slacks for dynamic and leakage energy reduction, in: MICRO 2001, 2001.
- [56] Y. Zhang, X. Hu, D.Z. Chen, Task scheduling and voltage selection for energy minimization, in: Design Automation Conference (DAC), 2002.
- [57] T. Zhou, X. Hu, E.H.-M. Sha, Estimating probabilistic timing performance for real-time embedded systems, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 9 (6) (2001).
- [58] Y. Zhu, G. Magklis, M.L. Scott, C. Ding, D.H. Albonese, The energy impact of aggressive loop fusion, in: IEEE PACT, 2004.
- [59] Q. Zhuge, E.H.-M. Sha, B. Xiao, C. Chantrapornchai, Efficient variable partitioning and scheduling for dsp processors with multiple memory modules, IEEE Trans. Signal Process. 52 (4) (2004).
- [60] Q. Zhuge, B. Xiao, E.H.-M. Sha, Code size reduction technique and implementation for software-pipelined dsp applications, ACM Trans. Embedded Comput. Systems 2 (4) (2003).



Meikang Qiu received the B.E. and M.E. degrees from Shanghai Jiao Tong University, China. He received the M.S. and Ph.D. degrees of Computer Science from University of Texas at Dallas in 2003 and 2007, respectively. From August 2007, he is an assistant professor of Electrical Engineering at University of New Orleans. He has worked at Chinese Helicopter R&D Institute, IBM HSPC, and Shenzhen Quality and Technology Supervision Bureau. He is an IEEE Senior member. He now serves as the vice general chair of the 2008 IEEE CSE 2008. He has been session chair and TPC

member for many international conferences, such as PDCS 2007, IEEE SEC 2008, and GlobeCom 2008. His research interests include embedded systems, computer and network security, and wireless sensor networks.

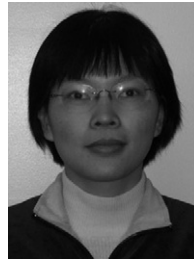


Edwin Hsing-Mean Sha received the M.A. and Ph.D. degree from the Department of Computer Science, Princeton University, Princeton, NJ, in 1991 and 1992, respectively. From August 1992 to August 2000, he was with the Department of Computer Science and Engineering at University of Notre Dame, Notre Dame, IN. He served as Associate Chair from 1995 to 2000. Since 2000, he has been a tenured full professor in the Department of Computer Science at the University of Texas at Dallas. He has published more than 220 research papers in refereed conferences and journals. He has served as an editor

for many journals, and as program committee member and Chair for numerous international conferences. He received Oak Ridge Association Junior Faculty Enhancement Award, Teaching Award, Microsoft Trustworthy Computing Curriculum Award, NSF CAREER Award, and NSFC Overseas Distinguished Young Scholar (B).



Meilin Liu joined the CS&E Department as a faculty member at Wright State University at Fall 2006. She received her Ph.D. degree in Computer Science from University of Texas at Dallas in 2006 and a Master's degree in 2004, and she received her Master's Degree in Control Theory & Control Engineering from Hohai University in 2000. Her research interests include embedded systems, computer architecture, optimizing compiler for specific architectures, and loop transformation techniques.



Man Lin received the B.E. degree in Computer Science and Technology from Tsinghua University, China, 1994. She received the Lic. and Ph.D. degrees from the Department of Computer Science and Information at Linkopings University, Sweden, in 1997 and 2000, respectively. She is currently an Associate Professor in Computer Science at St. Francis Xavier University, Canada. Her research interests include real-time and embedded system design and analysis, scheduling, power aware computing, and optimization algorithms.



Shaoxiong Hua received the B.S. and M.S. degrees in Instrument Science from Zhejiang University, Hangzhou, China, in 1992 and 1995, respectively. He received the M.S. and Ph.D. degrees in Computer Engineering from University of Maryland, College Park, MD, in 2003 and 2004, respectively. Since 2004, he has been a senior R&D engineer in Synopsys Inc., Mountain View, CA. From 1995 to 1998, he was an Assistant Professor in the Department of Scientific Instruments, Zhejiang University. His research interests include embedded/real-time

systems, electronic design automation, hardware/software co-design and low power system design.



Laurence T. Yang is a Professor in Computer Science at St Francis Xavier university, Canada. His research includes high performance computing and networking, embedded systems, ubiquitous/pervasive computing and intelligence. He has published more than 200 papers in refereed journals, conference proceedings, and book chapters in these areas. He has been involved in numerous conferences and workshops as a program/general conference chair and as a program committee member. In addition, he is the editor-in-chief of several international journals and few book series. He is

serving as an editor for around 20 international journals and acting as an author/co-author or an editor/co-editor of 30 books from Kluwer, Springer, Nova Science, American Scientific Publishers and John Wiley & Sons. He has won four Best Paper Awards, one IEEE Best Paper Award, 2007; one IEEE

Outstanding Paper Award, 2007; three Best Paper Nominations; Distinguished Achievement Award, 2005; Distinguished Contribution Award, 2004; Outstanding Achievement Award, 2002; and Canada Foundation for Innovation Award, 2003.