Multi-level Loop Fusion with Mimimal Code Size *

Meilin Liu, Zili Shao, Chun Xue, Kevin F. Chen, Edwin H.-M. Sha
Department of Computer Science
University of Texas at Dallas
Richardson, Texas 75083, USA
{mxl024100, zxs015000, cxx016000, fxc015200, edsha}@utdallas.edu

Abstract

Loop fusion is one of the most effective and common techniques to increase the performance of programs with multiple loops. The code size of the fused loop after loop fusion legalization might be increased due to the generation of the extra prologue and epilogue. Code size should be considered and minimized for embedded system designs. In this paper, we develop the Select_LF technique, which is to select one of the possible dimensions to legalize loop fusion such that the resultant code size of the fused loop is minimized. We propose a theoretical foundation to determine which dimensions are possible to be retimed to legalize loop fusion for any "J+K" model loop with a J-level outer loop and multiple K-level inner loops. The experimental results show that the execution time of the fused loops by our Select_LF technique is reduced by 55.4% on average compared to the original loops, and the code size of the fused loops is reduced by 7.8% on average compared to the previously reported Max_LF algorithm.

Keywords: Loop Fusion, Retiming, Code Size

1 Introduction

Many embedded processors such as TMS320C6x DSP (Digital Signal Processing) processor, Philip TriMedia processor, etc., use VLIW (Very Long Instruction Word) architecture that have multiple functional units to satisfy the high performance requirements. Since loops are prevalent in multimedia processing, digital signal processing, etc., loop optimization techniques are needed such that we can fully exploit the multiple-functional-unit architecture to improve the performance. Loop fusion is one of the most effective and common techniques to increase the performance of programs by combining and transforming multiple loops into one loop. However, loop fusion is not always applicable because of the existence of conflict data dependences among loops. Although the previous techniques such as loop shifting or retiming can solve this problem, the code size of the fused loop by these techniques might be increased significantly because of the generation of the prologue and epilogue. For embedded systems, code size is an important factor because of tight memory constraint. Therefore, it becomes an important problem to develop an effective loop fusion technique with minimal code size.

A lot of research has been done in loop fusion to improve the instruction-level parallelism and enhance data locality [1, 3, 6, 8]. The above loop fusion techniques do not consider the resultant code size of a fused loop which is another critical concern for embedded system design [2, 9]. In [5], we proposed a loop fusion technique, the maximum loop fusion technique (Max_LF), that can maximize the opportunities of loop fusion with the consideration of the resultant code size. However, the Max_LF technique might not achieve the minimal code size since it always retimes the $(J+1)$-th dimension to legalize loop fusion for any "J+K" model loop with a J-level outer loop and multiple K-level inner loops. In this paper, we propose the Select_LF technique that can select one of the possible dimensions to legalize loop fusion such that the resultant code size of the fused loop is minimal. In the following, we show an example of loop fusion problem.

The code shown in Fig. 1(a) contains two sequential loops enclosed in one shared outermost loop. To improve the performance of this program, one of the solutions is to fuse all the loops. But these two loops cannot be fused directly. The computation of $B[i,j,k]$ in the second loop requires the value of $A[i,j,k+1]$ in the first loop, while $A[i,j,k+1]$ has not been produced in (i,j,k)-th iteration, if we directly merge the loops. This kind of data dependence is called fusion-preventing dependence. We use the loop dependence graph (LDG) and the graph transformation technique based on the retiming concept proposed in [5] to legalize loop fusion in this paper. For the loops shown in Fig. 1(a), we can retime one dimension, either the second or the third dimension to legalize loop fusion. By retiming loops *L1* and *L2*, all the fusion-preventing dependences, can be eliminated. The Max_LF technique retimes the second dimension to legalize loop fusion, and the fused loop is shown in Fig. 1(b). We found that we can also retime the third dimension to legalize loop fusion, and the fused loop is shown in Fig. 1(c).

The code size of the fused loop by retiming the second dimension as shown in Fig. 1(b) is 14 instructions, and the code size of the fused loop by retiming the third dimension as shown in Fig. 1(c) is 12 instructions. The

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

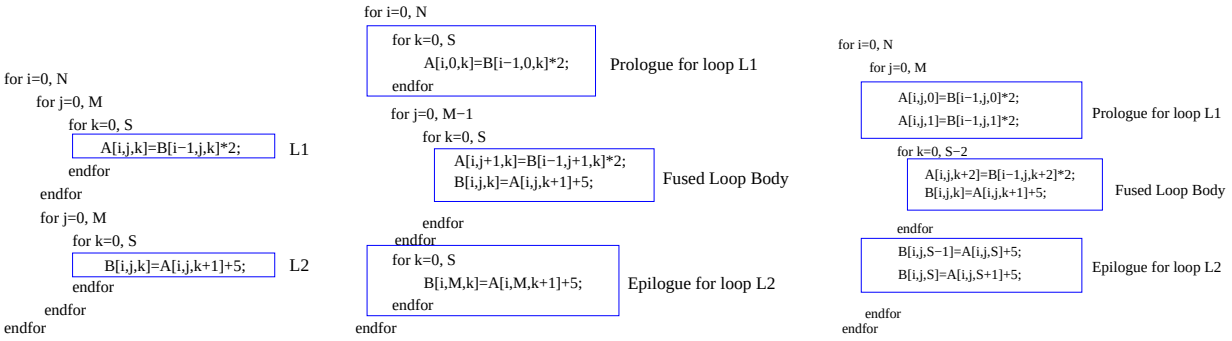


Figure 1: (a) The original 3-level loop with two inner loops. (b) The fused loop by retiming the second dimension. (c) The fused loop by retiming the third dimension.

execution times of the two fused loops by retiming the second dimension and by retiming the third dimension are about the same without considering the difference of the execution time for the prologue and epilogue. Compared to the original loop, it is reduced about 50%, assuming that any computation can be finished within one time unit, and there are 8 functional units. We propose the Select_LF technique to do the exclusive search to select one dimension to legalize loop fusion such that the code size of the fused loop is minimal. For the example shown in Fig. 1(a), our Select_LF technique selects to retime the third dimension to legalize loop fusion since it achieves a smaller code size.

In this paper, we first propose a theoretical foundation to determine which dimensions are possible to be retimed to legalize loop fusion for any "J+K" model loop. Then, we propose the Select_LF technique to select one dimension to legalize loop fusion such that the resultant code size of the fused loop is minimal. The experimental results show that the execution time of the fused loops by our Select_LF technique is reduced by 55.4% on average compared to the original loops, and the code size of the fused loops is reduced by 7.8% on average compared to the previously reported Max_LF algorithm.

The rest of this paper is organized as follows: We introduce the basic concepts and principles related to our technique in Section 2. In Section 3, we show that which dimensions are possible to be retimed to legalize loop fusion, and propose the Select_LF technique. Section 4 presents the experimental results. Section 5 concludes the paper.

2 Basic Concepts

In this section, we provide an overview of the basic concepts and principles related to our loop distribution technique.

2.1 Loop Dependence Graph

Loop dependence graph is a higher-level graph model compared to the data flow graph [5]. It is used to model the data dependences between multiple loops. A multi-dimensional loop dependence graph (MLDG) $G = (V, E, \delta, o)$ is a node-labeled and edge-weighted directed graph, where V is a set of nodes representing the loops. $E \subseteq V \times V$ is a set of edges representing data dependences between the loops. δ is a function from E to Z^n representing the minimum data dependence vector between the computations of two loops. o is a function from V to positive integers, representing the order of the execution sequence. All the comparisons between two loop dependence vectors are based on the lexicographic order in this paper. A N-dimensional vector $\vec{v}$ is smaller than a N-dimensional vector $\vec{u}$ according to the lexicographic order if and only if $v_1 = u_1, \cdots, v_{l-1} = u_{l-1}$ and $v_l < u_l$ for some integer $l \leq N$.

The loop dependence graph of the loop in Fig. 2(a) is shown in Fig. 2(b). There are three nodes $V = \{L1, L2, L3\}$ in the loop dependence graph which represent the three innermost loops in the program. The loop dependence edges are $E = \{e_1 : L1 \rightarrow L2,\ e_2 : L1 \rightarrow L3,\ e_3 : L2 \rightarrow L3,\ e_4 : L3 \rightarrow L2,\ e_5 : L3 \rightarrow L1\}$. The loop dependence vectors are $\{(0,-1),(0,0)\}$ between nodes $L1$ and $L2$, $\{(0,-2),(0,-1)\}$ between nodes $L1$ and $L3$, $\{(0,0)\}$ between nodes $L2$ and $L3$, $\{(1,0)\}$ between nodes $L3$ and $L2$, and $\{(2,-1)\}$ between nodes $L3$ and $L1$. According to our MLDG definition, $\delta(e_1) = (0,-1)$, $\delta(e_2) = (0,-2)$, $\delta(e_3) = (0,0)$, $\delta(e_4) = (1,0)$, $\delta(e_5) = (2,-1)$.

In a loop dependence graph, a fusion-preventing dependence is represented by an edge e with edge weight $\delta(e) < (0,0,\cdots,0)$. The fusion-preventing dependence edges for the LDG shown in Fig. 2(b) are e_1 and e_2.

2.2 Multi-Dimensional Retiming

Leiserson and Saxe [4] proposed the retiming technique. Passos et. al. [7] developed the multi-dimensional retiming technique to optimize the multi-dimensional problems. Retiming redistributes delays, i.e., data dependence distances, in a graph to achieve the minimum cycle time. Note that the retiming technique preserves all the data dependences of the original LDG. In this paper, we develop a graph transformation technique to remove fusion-preventing dependences based on the multi-dimensional retiming technique.

For a multi-dimensional loop dependence graph G, a multi-dimensional 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

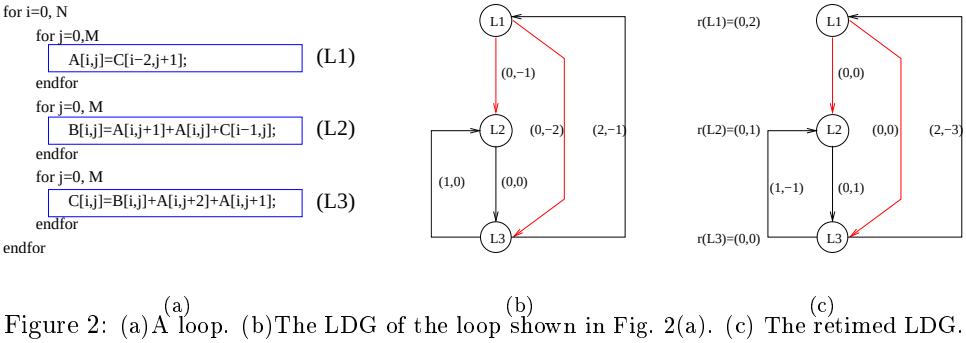


Figure 2: (a)A loop. (b)The LDG of the loop shown in Fig. 2(a). (c) The retimed LDG.

$w \rightarrow u$, 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 loop dependence graph, 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 [9].

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 [9]. From this definition, we know that the normalized retiming value for any node u in V is larger than or equal to $(0, 0)$ in the two-dimensional case. For example, the retiming values for nodes $L1, L2$ and $L3$ of the LDG in Figure 2(b) 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 $\vec{r}(L1) = (0, 0) - (0, -2) = (0, 2)$, $\vec{r}(L2) = (0, -1) - (0, -2) = (0, 1)$, and $\vec{r}(L3) = (0, -2) - (0, -2) = (0, 0)$. We retime the three nodes of the LDG in Figure 2(b) with the normalized retiming values $\vec{r}(L1) = (0, 2)$, $\vec{r}(L2) = (0, 1)$, and $\vec{r}(L3) = (0, 0)$. The retimed MLDG G^r of the LDG in Figure 2(b) is shown in Figure 2(c). After retiming, the weight of edge e_1 becomes $\delta^r(e_1) = (0, -1) + (0, 2) - (0, 1) = (0, 0)$. And the weight of edge e_2 becomes $\delta^r(e_2) = (0, -2) + (0, 2) - (0, 0) = (0, 0)$. Therefore, all the fusion-preventing dependences are removed.

2.3 Loop Properties

We propose a "J+K" model to represent any nested loop that contain sequential K-level loops enclosed within the J-level shared outer loop. It is obvious that $J \geq 0$ and $K \geq 1$. The loop levels are numbered from the outermost loop level to the innermost loop level. The loop level of the outermost loop level for the "J+K" model loop is 1, and the loop level of the innermost loop level for the "J+K" model loop is $(J + K)$.

In the "J+K" model, when $J \geq 1$, several K-level loops are enclosed inside the shared J-level outer loop. There may exist some loop-carried data dependences in this case, which may cause a dependence cycle in the corresponding *LDG*. For a nested loop in "J+K " model ($J \geq 1$) to be legally executed, each cycle in the LDG must contain at least one edge e representing an outer loop-carried dependences by definition, whose weight $\delta(e)$ has positive value on the first J dimensions, i.e., $(\delta_1(e), \delta_2(e), \cdots, \delta_J(e)) > (0, 0, \cdots, 0)$. Otherwise, if $(\delta_1(e), \delta_2(e), \cdots, \delta_J(e))$ is a negative value, i.e., $(\delta_1(e), \delta_2(e), \cdots, \delta_J(e)) < (0, 0, \cdots, 0)$, a computation in the loop will depend on some values produced in a future iteration. Then, the loop becomes illegal. The property stated in Property 2.1 ensures the legality of "J+K" model loop and also provides an important condition for applying our legalizing fusion algorithms.

Property 2.1 *A Loop Dependence Graph $G = (V, E, \delta, o)$ of a "J+K" model loop (where $J \geq 1$) is a legal LDG if the value of the first J-dimensions of an edge weight $\delta(e)$ satisfies that $(\delta_1(e), \delta_2(e), \cdots, \delta_J(e)) \geq (0, 0, \cdots, 0)$, $\forall e \in E$, and for any cycle $c = \{v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_n \rightarrow v_1\}$, the summation of the edge weight $\delta(c)$ satisfies that $(\delta_1(c), \delta_2(c), \cdots, \delta_J(c)) > (0, 0, \cdots, 0)$, where $\delta_i(e)$ denotes the i-th element of the vector $\delta(e)$.*

3 Legalizing Loop Fusion with Minimal Code Size

In [5], we showed that that we can always retime $(J + 1)$-th dimension to legalize loop fusion for a "J+K" loop, and we proposed the maximum loop fusion technique (Max_LF) to achieve the goal. In this section, we show that there are other possible dimensions we can retime so the resultant code size can be reduced. We first show which dimensions are possible to be retimed to legalize loop fusion. Then we propose the Select_LF technique to select one of the possible dimensions to legalize loop fusion such that the code size of the fused loop is minimal.

3.1 Possible Dimensions for Legalizing Loop Fusion

In this section, we provide a theorem to show which dimensions are possible to be retimed to legalize loop fu-

sion. We first present some necessary definitions, and then propose the theorem.

Before we present the theorem for the legalizing loop fusion, we need to define the minimum preventing dimension (Min_Prev) of an LDG. First, we define the preventing-dimension ($Prev\_Dim$) of an edge weight $\delta(e)$. If the edge weight $\delta(e)$ of an edge e in the LDG of a "J+K" model loop satisfies that $(\delta_1(e), \delta_2(e), \cdots, \delta_{J+K}(e)) \geq (0, \cdots, 0)$, then we define the preventing-dimension $Prev\_Dim$ of the edge weight $\delta(e)$ to be $Prev\_Dim(\delta(e)) = \infty$. Otherwise, if an edge e has a negative edge weight, i.e., $(\delta_1(e), \delta_2(e), \cdots, \delta_{J+K}(e)) < (0, \cdots, 0)$, then we define the preventing-dimension $Prev\_Dim$ of $\delta(e)$ as $Prev\_Dim(\delta(e)) = i$, such that $(\delta_1(e), ..., \delta_{i-1}(e)) = (0, \cdots, 0)$, and $\delta_i(e) < 0$, i.e., $Prev\_Dim(\delta(e))$ is the minimum dimension of the edge weight vector $\delta(e)$ that gives a negative value. For example, for an edge e with negative edge weight $\delta(e) = (0, -1, 0)$, the first negative element is in the second dimension, so $Prev\_Dim(\delta(e)) = 2$, and for an edge with positive edge weight $\delta(e) = (0, 1, 0), Prev\_Dim(\delta(e)) = \infty$.

Then, we define the minimum preventing dimension (Min_Prev) of a LDG to be the minimum dimension to give the negative value by all the edge weights of the LDG, i.e., $Min\_Prev = min_e\{\ Prev\_Dim(\delta(e))\ \}, \forall e$. From the definition, we can see that all the edges e with edge weight $\delta(e) \geq (0, \cdots, 0)$ must have $Prev\_Dim(\delta(e)) = \infty$. Only the preventing dependence edges e with $\delta(e) < (0, \cdots, 0)$ have the property that $J+1 \leq Prev\_Dim(\delta(e)) \leq J+K$ for any "J+K" model loop. The computed minimum preventing dimension (Min_Prev) is the maximum dimension that we can retime to eliminate the preventing dependences.

In the following, we show how to compute the preventing dimension ($Prev\_Dim$) of the edge weights and the minimum preventing dimension (Min_Prev) of the LDG shown in Fig. 3(a). For example, for edge $\{e1 : L1 \rightarrow L2\}$, $\delta(e1) = (0, 0, -1)$, the first negative element of the negative edge weight $(0, 0, -1)$ is in the third dimension, so $Prev\_Dim(\delta(e1)) = 3$. For edge $\{e2 : L2 \rightarrow L3\}$, $\delta(e2) = (0, 0, 3)$, $(0, 0, 3$ is a positive edge weight, so $Prev\_Dim(\delta(e2)) = \infty$. We take the minimum value of all the $Prev\_Dim$ as the minimum preventing dimension, so $Min\_Prev = 3$.

Based on the loop properties, we propose that we can retime one of the possible dimensions from $(J+1)$-th dimension to Min_Prev-th dimension to legalize the loop fusion for a "J+K" loop as stated in theorem 3.1.

Theorem 3.1 *Given a "J+K" model loop and its corresponding loop dependence graph $G = (V, E, \delta, o)$, we can retime one of the possible dimensions from the $(J+1)$-th dimension to Min_Prev-th dimension to legalize loop fusion, where $J + 1 \leq Min\_Prev \leq J + K$.*

For example, the LDG shown in Fig 3(a) has $J = 1$ and $Min\_Prev = 3$ according to the definition. According to theorem 3.1, we can retime one of the possible dimensions from $J + 1$ to Min_Prev to legalize loop fusion. So we can retime either the second or the third dimension of the LDG to legalize loop fusion.

3.2 The Select_LF Technique

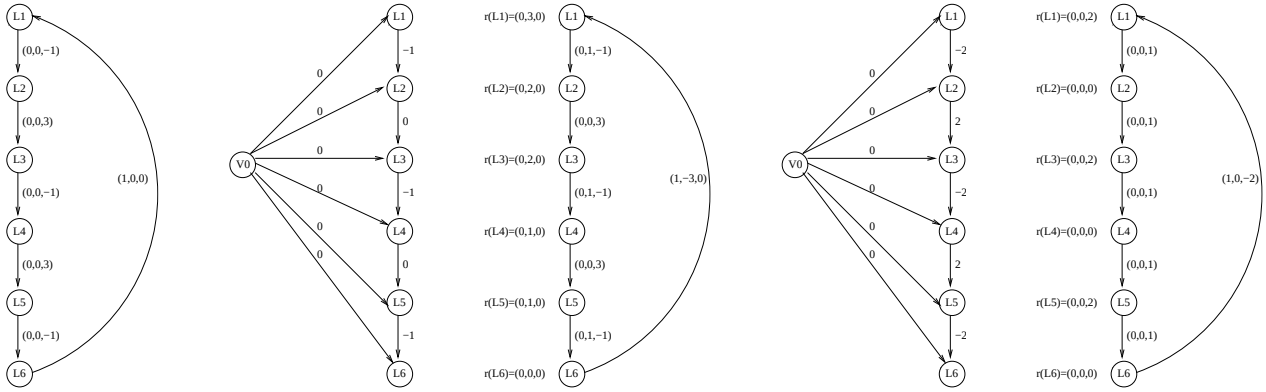
According to theorem 3.1, we can retime one of the possible dimensions from $J + 1$ to Min_Prev to legalize loop fusion. We propose the Select_LF technique as shown in algorithm 3.1 to search and select one of the possible dimensions from $J+1$ to Min_Prev to legalize loop fusion such that the resultant code size of the fused loop is minimal. When $Min\_Prev = (J + 1)$, then we can only retime the $(J + 1)$-th dimension.

Algorithm 3.1 Selecting Dimension to Legalize Loop Fusion (Select_LF)

Require: A loop dependence graph $G = (V, E, \delta, o)$
Ensure: Dimension number dim
  /* Compute the minimum preventing dimension */
  $Min\_Prev = min_e\{\ Prev\_Dim(\delta(e))\ \}, \forall e$
  $dim = J + 1, size = \infty$
  for $i \leftarrow J + 1$ to Min_Prev **do**
    Compute the set of retiming values R for dimension i to legalize loop fusion by the shortest path algorithm [5]
    Compute the maximum retiming value R_i for i-th dimension
    Compute the code size C_i of the fused loop using the retiming values for i-th dimension
    if $C_i < size$ **then**
      $dim \leftarrow i$
    end if
  end for
  return dim

In the algorithm 3.1, we computed the retiming values using the shortest path algorithm for all the possible dimensions from $(J+1)$-th dimension to Min_Prev-th dimension. Based on the retiming values computed for each dimension, we compute the code size of the fused loop using the code-size calculation formula proposed in [5]. Then, we select the dimension to legalize loop fusion such that the code size of the fused loop is minimal.

For example, for the LDG shown in Fig 3(a), since $J = 1$, $Min\_Prev = 3$, we can retime either the second or the third dimension to legalize loop fusion according to theorem 3.1. The Max_LF technique retimes the second dimension to legalize loop fusion. The constraint graph for computing the retiming values for the second dimension is shown in Fig 3(b), and the normalized retiming values computed by the shortest path algorithm for the second dimension are $r(L1) = (0, 3, 0), r(L2) = (0, 2, 0), r(L3) = (0, 2, 0), r(L4) = (0, 1, 0), r(L5) = (0, 1, 0), r(L6) = (0, 0, 0)$. Let R_i be the maximum retiming value for i-th dimension, i.e., $R_i = \max_u\{r_i(u)\}$. The maximum retiming values for the second dimension is $R_2 = 3$. Similarly, the constraint graph for computing the retiming values for the third dimension is shown in Fig 3(d). And the normalized retiming values for the third dimension are $r(L1) = (0, 0, 2), r(L2) = (0, 0, 0), r(L3) = (0, 0, 2), r(L4) = (0, 0, 0), r(L5) = (0, 0, 2), r(L6) = (0, 0, 0)$. The maximum retiming value for the third dimension is $R_3 = 2$. The maximum retiming value for the third dimension is less than



(a) (b) (c) (d) (e)

Figure 3: (a)The LDG of a loop. (b) The constraint graph for retiming the second dimension. (c) The retimed LDG by retiming the second dimension. (d) The constraint graph for retiming the third dimension. (e) The retimed LDG by retiming the third dimension.

the maximum retiming value for the second dimension, i.e., $R_3 < R_2$. Also the code size of the fused loop by retiming the third dimension is less than the code size of the fused loop by retiming the second dimension.

The Select_LF technique as shown in algorithm 3.1 does the exclusive search to select one dimension to legalize loop fusion such that the resultant code size of the fused loop is minimal. For the example LDG shown in Fig. 3(a), the dimension selected by the algorithm 3.1 will be 3, since the code size of the fused loop by retiming the third dimension is less than the code size of the fused loop by retiming the second dimension. Compared to the number of the loop units, the dimension of the edge weight is a small number. So our algorithm is polynomial time with respect to the number of the loop units (the number of the nodes in the LDG).

4 Experiments

This section presents the experimental results of our technique. We simulate a DSP processor with eight functional units and compare the execution time of the original loop and the fused loop by the Max_LF technique and the Select_LF technique. We also compare the code sizes of the fused loop by the Max_LF technique and the Select_LF technique. The standard list scheduling algorithm is used in our experiments. All the loop dependence graphs used in our experiments have fusion-preventing dependences and some of them are cyclic. In our experiments, all the experimental cases are from the 3-level nested loops. CASE1, CASE2, CASE3, CASE4 and CASE5 refer to the LDGs presented in Fig. 5(a), Fig. 9(a), Fig. 12(a), Fig. 12(b) and Fig. 12(c) in paper [5]. CASE6 refers to the LDG shown in Fig. 3(a). The nodes of an LDG are obtained from the DSP benchmarks.

The experimental results are provided in Table 1 and Table 2. Table 1 compares the execution time of the original loop and the fused loop by the Max_LF technique and the Select_LF technique when there are 8 Functional Units. The Column "*Original*" contains one field: the execution time of the original loop (Time). The Column "*Max_LF* " contains two fields: the execution time of the fused loop by the Max_LF technique (Time), and the improvement of the execution time of the fused loop by the Max_LF technique compared to the original loop (Time Impro.). The Column "*Select_LF* " contains two fields: the execution time of the fused loop by the Select_LF technique (Time), and the improvement of the execution time of the fused loop by the Select_LF technique compared to the original loop (Time Impro.). The execution time is defined to be the schedule length times the total iterations. The schedule length is the number of time units to finish one iteration of the loop body. For the sequentially executed loops, the execution time is the sum of the execution time of each individual loop. In Table 1, W represents the total numbers of the iterations, and $W = N * M * S$. Here, N is the total numbers of the iterations for the outermost loop, M is the total numbers of the iterations for the second-level loop, and S is the total numbers of the iterations for the innermost loop. From Table 1, we can see that the fused loop by the Max_LF technique achieves the same timing performance with the fused loop by the Select_LF technique since both of these two techniques fully exploit the instruction-level parallelism.

Table 2 compares the code sizes of the fused loop by the Max_LF technique and our Select_LF technique. The Column "*Max_LF* " contains two fields: the dimension retimed by the Max_LF technique to legalize loop fusion (Dim-Retimed), the code size of the fused loop by the Max_LF technique (Size). The Column "*Select_LF* " contains three fields: the dimension selected by the algorithm 3.1 to legalize loop fusion (Dim-Selected), the code size of the fused loop by the Select_LF technique (Size), and the improvement of the code size of the fused loop by the Select_LF technique compared to the fused loop by the Max_LF technique (Size Impro.). For CASE1, CASE3 and CASE4, CASE5, both the Max_LF technique and the Select_LF retime the same dimension to legalize loop fusion. Our Select_LF technique has no choice and retimes the $(J+1)$-th dimension to legalize loop fusion since $Min\_Prev = J+1$. This is why the code sizes of the fused

Program	Original	Max_LF		Select_LF	
	Time	**Time**	**Time Impro.**	**Time**	**Time Impro.**
CASE1	11*W	7*W	45.4%	7*W	45.4%
CASE2	24*W	9*W	62.5%	9*W	62.5%
CASE3	34*W	12*W	64.7%	12*W	64.7%
CASE4	15*W	7*W	53.3%	7*W	53.3%
CASE5	28*W	9*W	67.9%	9*W	67.9%
CASE6	39*W	24*W	38.4%	24*W	38.4%
Improvement	-	-	55.4%	-	55.4%

Table 1: The execution time of the original loop and the fused loops by the Max_LF technique and the Select_LF technique.

Program	Max_LF		Select_LF		
	Dim-Retimed	**Size**	**Dim-Selected**	**Size**	**Size Impro.**
CASE1	2	100	2	100	0%
CASE2	2	154	3	194	25.9%
CASE3	2	108	2	108	0%
CASE4	2	240	2	240	0%
CASE5	3	166	3	166	0%
CASE6	2	194	3	234	20.6%
Improvement	-		-		7.8%

Table 2: The code sizes of the fused loop by the Max_LF technique and the Select_LF technique.

loop by our Select_LF technique have no improvement compared to the code sizes of the fused loop by the Max_LF technique for these test cases. For CASE2 and CASE6, the Max_LF technique retimes the second dimension to legalize loop fusion, and the Select_LF selects the third dimension to legalize loop fusion. When the Select_LF technique selects a different dimension to legalize loop fusion, it achieves a smaller code size. For example, for CASE6, the Max_LF technique retimes the second dimension to legalize loop fusion, but our Select_LF technique selects the third dimension to legalize loop fusion. The code size of the fused loop by our Select_LF technique is 194, while the code size of the fused loop by the Max_LF technique is 234. The experimental results show that the execution time of the fused loops by our Select_LF technique is reduced 55.4% on average compared to the original loop. The code size of the fused loop by the Select_LF technique is reduced 7.8% on average compared to the code size of the fused loop by the Max_LF technique.

5 Conclusion

The maximum loop fusion technique (Max_LF) proposed in [5] can maximize the opportunities of loop fusion, but it cannot always achieve the minimal code size of the fused loop. In this paper, we further developed the Select_LF technique to select one dimension to legalize loop fusion such that the resultant code size of the fused loop is minimal. The experiments showed that the improved loop fusion technique proposed in this paper can achieve the same timing performance with the Max_LF technique, and the code size of the fused loop is smaller.

References

[1] R. Allen and K. Kennedy. *Optimizing Compilers for Modern Architectures: A Dependence-based Approach*. Morgan Kaufmann, 2001.

[2] E. Granston, R. Scales, E. Stotzer, A. Ward, and J. Zbiciak. Controlling code size of software-pipelined loops on the TMS320C6000 VLIW DSP architecture. In *Proc. of the 3rd IEEE/ACM Workshop on Media and Streaming Processors*, pages 29–38, Dec. 2001.

[3] K. Kennedy and 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, Number 768*, pages 301–320, 1993.

[4] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algortithmica*, 6(1):5–35, June 1991.

[5] M. Liu, Q. Zhuge, Z. Shao, and E. H.-M. Sha. General loop fusion technique for nested loops considering timing and code size. In *Proc. ACM/IEEE International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES 2004)*, pages 190–201, Sep. 2004.

[6] K. S. McKinley, S. Carr, and C.-W. Tseng. Improving data locality with loop transformations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 18(4):424 – 453, July 1996.

[7] N. L. Passos and E. H.-M. Sha. Achieving full parallelism using multi-dimensional retiming. *IEEE Transactions on Parallel and Distributed System*, 7(11):1150–1163, Nov. 1996.

[8] M. Wolfe. *High Performance Compilers for Parallel Computing*. Addison-Wesley Publishing Company, Inc., 1996.

[9] Q. Zhuge, B. Xiao, and E.-M. Sha. Code size reduction technique and implementation for software-pipelined DSP applications. *ACM Transactions on Embedded Computing Systems(TECS)*, 2(4):590–613, Nov. 2003.