

Loop Fusion via Retiming for DSP Applications *

Meilin Liu, Qingfeng Zhuge, Zili Shao, Kevin F. Chen, Edwin H.-M. Sha

Department of Computer Science

University of Texas at Dallas

Richardson, Texas 75083, USA

{mxl024100, qfzhuge, zxs015000, fxc015200, edsha}@utdallas.edu

Abstract

For DSP applications with multiple sequential loops or nested loops, loop fusion usually can be applied to increase the instruction-level parallelism. Loop fusion, however, is not always applicable because of the existence of fusion-prevention dependencies among loops. In this paper, we present an efficient loop fusion technique based on loop dependency graph model, retiming, and multi-dimensional retiming concepts. We show that any 1-level loop and 2-level nested loop can be legally fused by performing our legalizing fusion technique. Polynomial-time algorithms are developed to solve the loop fusion problem for both 1-level and 2-level loops. The experimental results show that our loop fusion technique always significantly reduces the schedule length.

Keywords: Loop Fusion, Retiming, DSP Processors

1 Introduction

DSP applications usually have strict timing requirement. For DSP processors with multiple functional units, such as TI's TMS320C6x, loop optimization techniques need to be employed to increase the parallelism and improve the utilization of functional units. Loop fusion is commonly used to improve the instruction-level parallelism by combining and transforming multiple loops into one loop [15, 11]. Many research works used loop fusion to enhance data locality and increase parallelism [6, 3, 7, 13, 1, 14]. Loop fusion, however, is restricted by fusion-prevention dependencies existing among loops. And most of the existing techniques cannot fuse the loops with fusion-prevention dependencies [4, 3, 11, 7, 13, 1]. The existing techniques that can handle fusion-prevention dependencies still have limitations. Therefore, efficient techniques for legalizing loop fusion are necessary for fully achieving the benefits of the loop fusion.

We use an example to show the loops with fusion prevention dependencies and how it can be transformed and

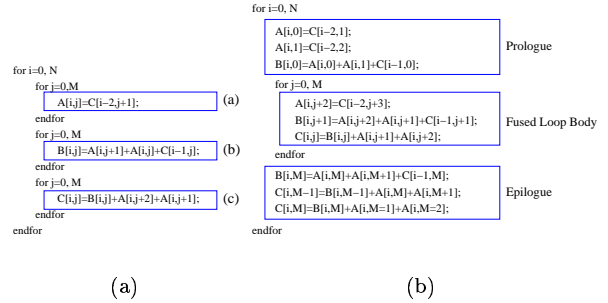


Figure 1: (a) The original 2-level loop with three inner loops. (b) The fused loop.

then legally fused. The example shown in Figure 1(a) contains three sequential loops enclosed in a shared outermost loop. In this simple example, we can find a fusion-prevention dependency between loop *a* and *b* by observation. Note that $B[i][j]$ is dependent on $A[i][j+1]$. If we directly fuse loop *a* and *b* into one loop, $A[i][j+1]$ will not be available in iteration (i, j) . Similarly, we can not directly fuse loop *a* and *c*, because $C[i][j]$ is dependent on $A[i][j+1]$ and $A[i][j+2]$ which are not available in the same iteration. Many previous works can not deal with fusion-prevention dependencies, and thus can not fuse loop *a*, *b* and *c* into one loop. Some of them can only fuse loop *b* and *c* [4, 3, 11, 7, 13]. While using our technique, all the fusion-prevention dependencies can be removed by a graph transformation technique based on the *multi-dimensional retiming concept* [9, 10], and the final code after fusing loop *a*, *b*, and *c* is shown in Figure 1(b). Assuming there are four functional units, the schedule length of the fused loop can be dramatically reduced from 9 to 4 control steps.

Manjikian and Abdelrahman proposed the “shift-and-peel” loop transformation technique [6]. But it requires uniform dependencies among the loops. Actually, the “shift-and-peel” technique can be regarded as a special case in our proposed loop fusion technique. Passos et. al. proposed a nested loop fusion technique in [12], which targets to achieve the fully parallel execution of the innermost loop body on parallel architectures. However, the execution sequence of the nested loop can be changed and the code

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

size may become very large, which limits its application in embedded systems [2].

In this paper, we present an efficient loop fusion technique based on graph transformation by using retiming and multi-dimensional retiming concepts. By casting a retiming view on the loop fusion problem, we derive loop fusion theorems based on the understanding of the properties of loops to be fused. We show that any 1-level, 2-level loop can always be legally fused when appropriate transformation is applied. Several efficient legalizing fusion algorithms are proposed to solve the loop fusion problem for both 1-level and 2-level loops. Our contributions are as follows:

1. We develop an efficient graph transformation technique to legalize loop fusion for 1-level, 2-level loops based on the loop dependency graph (LDG) model, retiming, and multi-dimensional retiming concepts[5, 10].
2. We derive theorems to show the correctness of the legalizing fusion process.
3. We prove that any 1-level, 2-level loops can be legally fused using our legalizing fusion technique.
4. Polynomial-time algorithms are proposed to solve the loop fusion problem.

The experimental results show that our loop fusion technique significantly reduces the schedule length with a reasonable code size. For example, the schedule length of the Livermore benchmark LL18 is reduced by more than 55%, while the three sequential loops of LL18 cannot be fused by most of the previous techniques[4, 3, 11, 7, 13]. With all the experiments we did, the schedule length of the fused loops using our LF_IP algorithm can be reduced by more than 57%.

The rest of the paper is organized as follows: We introduce the basic concepts and principles related to our loop fusion technique in Section 2. The basic properties of the various loop models, and the legalizing fusion algorithms and theorems are presented in Section 3. The experimental results are presented in Section 4. And finally, Section 5 concludes the paper.

2 Basic Concepts

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

2.1 Loops and Loop Dependency Graph

Loop fusion is a loop optimization technique which combines several sequential loops into one loop. In this paper, we develop a legalizing fusion technique for 2-level nested loops. The technique can also be applied to 1-level loop fusion. The 2-level nested loops include “1+1” model loops which contain several 1-level loops in one outermost loop, and “0+2” model loops which contain several sequentially executed 2-level loops, as shown in Figure 2(a) and 2(b), respectively.

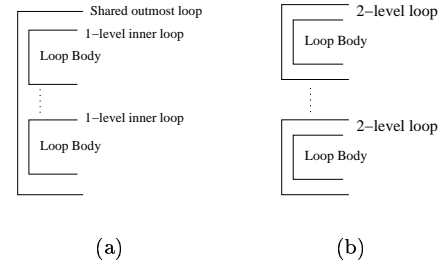


Figure 2: (a)“1+1” model. (b)“0+2” model.

Data dependency analysis is critical for loop fusion [15, 6, 8]. The fused loop should not violate any data dependency between the original loops which can be represented by the loop dependency vector[12]. 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 $(i_1, i_2, \dots, i_n) \in b$.

To clearly show the data dependencies between loops, we use a loop dependency graph (LDG) to model a nested loop. A multi-dimensional loop dependency graph (MLDG) $G = (V, E, \delta)$ is a node-weighted and edge-weighted directed graph, where V is a set of nodes representing the loops to be fused. $E \subseteq V \times V$ is a set of edges representing dependencies between the loops. δ is a function from E 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 as in [12] 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 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 $v_2 < u_2$.

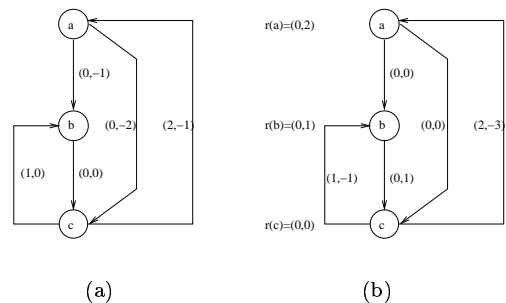


Figure 3: (a)The LDG of the loop shown in Figure 1(a). (b)The retimed LDG with retiming $\vec{r}(a) = (0, 2)$, $\vec{r}(b) = (0, 1)$, and $\vec{r}(c) = (0, 0)$.

In a loop dependency graph, a fusion-prevention dependency is represented by an edge e with edge weight $\delta(e) < (0, 0, \dots, 0)$. Fusion-prevention dependencies will become reverse to the control flow after the loop fusion is performed, making the fusion illegal [6, 8, 12]. The loop dependency graph of the loop in Figure 1(a) is shown in Figure 3(a). There are three nodes $V = \{a, b, c\}$ in the loop dependency graph which represent the three innermost loops in the program. The loop dependency edges are $E = \{e_1 : a \rightarrow b, e_2 : a \rightarrow c, e_3 : b \rightarrow c, e_4 : c \rightarrow b, e_5 : c \rightarrow a\}$. 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, $\{(1, 0)\}$ between nodes c and b, and $\{(2, -1)\}$ between nodes c and a. According to our MLDG definition, $\delta(e_1) = (0, -1)$, $\delta(e_2) = (0, -2)$, $\delta(e_3) = (0, 0)$, $\delta(e_4) = (1, 0)$, $\delta(e_5) = (2, -1)$. The fusion-prevention dependency edges are e_1 and e_2 .

2.2 Multi-Dimensional Retiming

Leiserson [5] proposed the retiming technique. Passos et. al. [10, 9] developed the multi-dimensional retiming technique to optimize the multi-dimensional problems. In this paper, we develop a graph transformation technique to remove fusion-prevention dependencies based on the multi-dimensional retiming technique.

For a multi-dimensional loop dependency 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 $w \rightarrow u$, 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. Note that the retiming technique preserves all the data dependencies of the original LDG. The retiming value $\vec{r}(u)$ means that a node u originally executed in the iteration i is moved to the iteration $i - \vec{r}(u)$. For a loop dependency 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 [16].

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 [16]. 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 computed by the legalizing fusion algorithm for nodes a, b and c of the LDG in Figure 3(a) are $(0, 0)$, $(0, -1)$, and $(0, -2)$ respectively. The minimum retiming value $\min_u r(u)$ is $(0, -2)$. 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 Figure 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 MLDG G^r of the LDG in Figure 3(a) is shown in Figure 3(b). 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-prevention dependencies are removed.

3 Legalizing Fusion Technique

The problem of legalizing fusion is to remove all the fusion-prevention dependencies in a loop dependency graph. We solve the problem with graph transformation technique based on multi-dimensional retiming concept. We found that all the fusion-prevention dependencies can be removed provided a legal MLDG. In this section, we first discuss the properties of the LDGs for various loop models which provide necessary conditions for the correctness of our legalizing fusion algorithms. Then, we propose two legalizing fusion algorithms for “1+1” model loop and extend them to deal with “0+1” and “0+2” model loops.

3.1 Basic Properties of Various Loop Models

For “0+1” or “0+2” model loops, the corresponding LDGs are directed acyclic graphs (DAGs). Therefore, there always exists a retiming or multi-dimensional retiming that can retime the negative weights to be non-negative according to the basic retiming concept [5, 10, 9]. Thus, all the fusion-prevention dependencies can be removed. Due to the space limitation, we do not provide the proof of the following lemma, but it can be easily proved by the basic retiming principles.

Lemma 3.1 *Any “0+1” and “0+2” model loop can be retimed so they can be legally fused after retiming.*

In the “1+1” model, several 1-level loops are enclosed inside the shared outermost loop. There may exist some loop-carried data dependencies in this case, which may cause a dependency cycle in the corresponding LDG. For a nested “1+1” model loop to be legally executed, each cycle in the LDG must contain at least one edge e representing an outermost loop-carried dependencies by definition, whose weight $\delta(e)$ has positive value on the first dimension, i.e., $\delta_1(e) \geq 1$. The property stated in Lemma 3.2 ensures the legality of “1+1” model loop and also provides an important condition for applying our legalizing fusion algorithms.

Lemma 3.2 *A Loop Dependency Graph $G = (V, E, \delta)$ of a “1+1” model loop is a legal LDG if the value of the first dimension of an edge weight $\delta(e)$ satisfies that $\delta_1(e) \geq 0$, $\forall e \in E$, and for any cycle $c = \{v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n \rightarrow v_1\}$, the summation of the edge weight $\delta(c)$ satisfies that $\delta_1(c) \geq 1$, where $\delta_1(e)$ denotes the first element of the vector $\delta(e)$.*

3.2 Legalizing Fusion Algorithms for “1+1” Model

In this subsection, we propose two legalizing fusion algorithms, the Basic_LF algorithm, and the LF_IP algorithm. The algorithms are developed to legalize fusion for “1+1” model loops and also can be extended to legalize fusion for “0+1” and “0+2” model loops. The Basic_LF algorithm transforms the graph without considering the critical path length of the fused loop. The LF_IP algorithm keeps the critical path length of the fused loop to be minimal when transforming the graph.

The Basic_LF Algorithm

Algorithm 3.1 Basic Legalizing Fusion Algorithm(Basic_LF)

Require: A loop dependency graph $G = (V, E, \delta)$
Ensure: A retiming function r of the loop dependency graph

```

/*remove all the edges  $e$  with  $\delta_1(e) \geq 1$  from  $E$  */
 $E' \leftarrow E - \{e \mid \delta_1(e) \geq 1\}$ 
Construct the constraint graph  $G_c = (V_c, E_c, w_c)$ , where
 $V_c \leftarrow V \cup \{v_0\}$ ,  $E_c \leftarrow E' \cup E_s$ , here  $E_s = \{(v_0 \rightarrow v_i) \mid v_i \in V\}$ 
for all edges  $e_c \in E'$  do
     $w_c(e_c) \leftarrow \delta_2(e_c)$ 
    /* In the LF_IP Algo.:  $w_c(e_c) \leftarrow \delta_2(e_c) - 1$  */
end for
for all edges  $e_c \in E_s$  do
     $w_c(e_c) \leftarrow 0$ 
end for
Call the Bellman-Ford algorithm to compute  $D(v_0, v_i)$ ,
the shortest distance from  $v_0$  to each node  $v_i \in V$ 
for all nodes  $v_i \in V$  do
     $r(v_i) = (0, D(v_0, v_i))$ 
end for
return  $r$ 

```

In the Basic_LF algorithm, we first remove all the edges e with $\delta_1(e) \geq 1$ from the LDG and we obtain a directed acyclic graph (DAG) G' . Then, we construct a constraint graph G_c of the shortest path problem by adding a source node v_0 and also the edges from v_0 to all the other nodes of G' . All the edges e belonging to the original LDG are labeled with $\delta_2(e)$ in the constraint graph G_c . The edges added from the source node to all the other nodes of G' are labeled with 0. The construction of the constraint graph G_c does not create cycles. The Bellman-Ford algorithm can be applied to compute the shortest distance from the source node v_0 to each other node v_i , which serves as the second element of the retiming value of node v_i . The first element of the retiming value of node v_i is set to be 0 in the Basic_LF algorithm. Actually, we apply retiming only on the second dimension. The graph transformation is then similar to a one-dimensional retiming. The complexity of the Basic_LF algorithm is $O(|V||E|)$.

An example of the graph transformation process of the Basic_LF algorithm is shown in Figure 4(a). The corre-

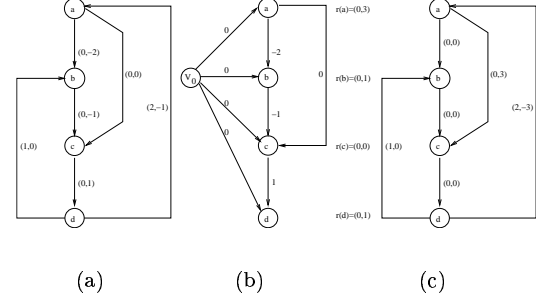


Figure 4: (a)Example LDG. (b) The constraint graph of the Basic_LF Alg. (c) The retimed LDG by the Basic_LF Alg.

sponding constraint graph is shown in Figure 4(b). The normalized retiming values computed by the Basic_LF algorithm are $\vec{r}(a) = (0, 3)$, $\vec{r}(b) = (0, 1)$, $\vec{r}(c) = (0, 0)$, $\vec{r}(d) = (0, 1)$. The retimed LDG is shown in Figure 4(c). There is no fusion-prevention dependency in the retimed LDG.

Theorem 3.1 Given a legal LDG of a “1+1” model loop, the Basic_LF algorithm can always transform the loops so that they can be fused legally.

Proof 3.1 According to lemma 3.2, every legal loop dependency graph $G = (V, E, \delta)$ must satisfy that $\delta_1(e) \geq 0, \forall e \in E$, and $\delta_1(c) \geq 1$, for each cycle c in G . Therefore, after all the edges e with $\delta_1(e) \geq 1$ are removed, the obtained graph G' is a DAG. The construction of the constraint graph G_c does not create cycles. Thus, G_c is a DAG. Bellman-Ford algorithm is applied to compute the single-source shortest distance $D(v_0, v_i)$ from v_0 to all the other nodes v_i , which is the second element of the retiming vector for node v_i , i.e., $r_2(v_i)$.

For an edge $e : v_i \rightarrow v_j$ labeled with $\delta_2(e)$ in the constraint graph, Bellman-Ford algorithm computes the shortest distance $D(v_0, v_i)$ from v_0 to v_i as the second element of the retiming vector for node v_i , i.e., $r_2(v_i)$, and the shortest distance $D(v_0, v_j)$ from v_0 to v_j as the second element of the retiming vector for node v_j , i.e., $r_2(v_j)$. It follows that,

$$r_2(v_j) \leq r_2(v_i) + \delta_2(e) \implies \delta_2(e) + r_2(v_i) - r_2(v_j) \geq 0$$

According to the definition of retiming, we have

$$\delta_2^r(e) = \delta_2(e) + r_2(v_i) - r_2(v_j) \implies \delta_2^r(e) \geq 0$$

In the Basic_LF algorithm, we only retime the second dimension, so $\delta_1^r(e) = \delta_1(e) \geq 0$. Therefore, we have $\delta^r(e) \geq (0, 0), \forall e \in E$, which means there is no fusion-prevention dependency in the retimed graph.

The LF_IP Algorithm

The Basic_LF algorithm retimes the LDG so that each edge e satisfies that $\delta^r(e) \geq (0, 0)$ in the retimed LDG G^r . But the critical path[10] of the fused loop based on the

Basic_LF algorithm can be increased when the zero-weight edge connecting the critical paths of two loops as shown in Figure 5(a). To increase the instruction level parallelism of the fused loop, we propose the LF_IP algorithm (Loop Fusion Algorithm with Instruction-Level Parallelism). In the LF_IP algorithm, the edge weight of the constructed constraint graph is computed differently. In the Basic_LF algorithm, the weight of the edges belonging to the original LDG is computed as $w_c(e) = \delta_2(e)$. In the LF_IP algorithm, however, the weight of the edges belonging to the original LDG is set as $w_c(e) = \delta_2(e) - 1$ in the constraint graph. This modification is important because it guarantees that any edge e in the retimed LDG satisfies that $\delta^r(e) \geq (0, 1)$. Therefore, the critical path of the fused loop is the longest critical path of the original loops as shown in Figure 5(b). In other words, the critical path of the fused loop will not be increased.

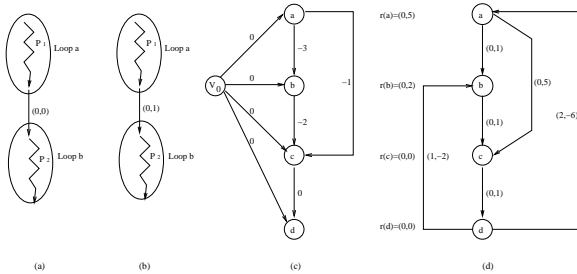


Figure 5: (a)The critical path of the fused loop by the Basic_LF Alg. (b)The critical path of the fused loop by the LF_IP Alg. (c)The constraint graph of the LF_IP Alg. (d)The retimed LDG by the LF_IP Alg.

Theorem 3.2 *Given a legal LDG of a “1+1” model loop, LF_IP algorithm can always transform the loops so that they can be legally fused and the critical path of the fused loop is minimal, i. e., the longest critical path of the original loops.*

We use the same example LDG shown in Figure 4(a) to show how the LF_IP algorithm works. The corresponding constraint graph for the LF_IP algorithm is shown in Figure 5(c). The normalized retiming values computed by the LF_IP algorithm are $\vec{r}(a) = (0, 5)$, $\vec{r}(b) = (0, 2)$, $\vec{r}(c) = (0, 0)$, $\vec{r}(d) = (0, 0)$. The retimed LDG is shown in Figure 5(d). Note that the longest zero-weight path of the LDG in Figure 4(c) is $a \rightarrow b \rightarrow c \rightarrow d$. While it just contains one node in Figure 5(d).

Legalizing Fusion Algorithms for “0+1” Model and “0+2” Model

Both the Basic_LF algorithm and the LF_IP algorithm can be easily extended to deal with “0+1” and “0+2” model loops. The loop dependency graphs of “0+1” and “0+2” model loops are DAGs. Therefore, we don’t need to break the cycles in a LDG as we do in the algorithms for “1+1” model loops.

The LDG of a “0+1” model loop is an one-dimensional graph. Thus, we can retime the LDG of a “0+1” model loop such that the edge weight of the retimed LDG satisfies that $\delta^r(e) \geq 0$, or $\delta^r(e) \geq 1$, $\forall e$.

For the “0+2” model loop, we consider the modifications on the legalizing fusion algorithms for “1+1” model one by one. For the “0+2” Basic_LF algorithm, the first dimension of the LDG is retimed to achieve the resultant LDG with $\delta^r(e) \geq (0, 0)$, $\forall e$. The “0+2” LF_IP algorithm is to retime the first dimension of the LDG of “0+2” model loop such that the edge weight of the resultant LDG satisfies $\delta^r(e) \geq (0, 1)$, $\forall e$.

4 Experiments

This section presents the experimental results of our algorithms. We simulate a DSP processor with 4, 8 function units and compare the schedule length of the original loop and the fused loop. The standard list scheduling algorithm is used in our experiments. All the loop dependency graphs used in our experiments have fusion-prevention dependencies. Most of the previous works cannot do the loop fusion in this situation or cannot fuse all the loops into one loop because of the fusion-prevention dependencies.

LL18 is the eighteenth kernel from the Livermore benchmark. LDG1, LDG2, and LDG3 refer to the examples presented in Figure 2, 8, and 17 in [12]. LDG4 is shown in Figure 3(a). LDG5 is shown in Figure 4(a). All these example LDGs are from the 2-level nested loops. The nodes of an LDG are obtained from the DSP benchmarks including WDF (Wave Digital filter), IIR (Infinite Impulse Response filter), DPCM (Differential Pulse-Code Modulation device), and 2D (Two Dimensional filter).

Program	Orig.	Basic_LF	%	LF_IP	%
LDG1	9	5	44.4%	4	55.6%
LDG2	37	18	51.3%	17	54.1%
LDG3	41	22	46.3%	22	46.3%
LDG4	23	12	47.8%	8	65.2%
LDG5	29	13	55.2%	9	68.9%
LL18	9	5	44.4%	4	55.6%
Average Improvement			48.2%	-	57.6%

Table 1: Schedule length of the original loop and the fused loop when there are 4 FUs

Program	Orig.	Basic_LF	%	LF_IP	%
LDG1	9	5	44.4%	3	66.7%
LDG2	36	10	72.2%	9	75.0%
LDG3	38	14	63.2%	11	71.1%
LDG4	23	12	47.8%	7	69.6%
LDG5	29	13	55.2%	7	75.9%
LL18	9	5	44.4%	4	55.6%
Average Improvement			54.5%	-	68.9%

Table 2: Schedule length of the original loop and the fused loop when there are 8 FUs

Table 1 displays the schedule length of the original loop and the fused loop when there are 4 function units in the

DSP processor. The second column shows the schedule length of the original loop. The third column shows the schedule length of the fused loop by the Basic_LF algorithm. The fourth column shows the improvement of the schedule length of the fused loop by the Basic_LF algorithm. The fifth column shows the schedule length of the fused loop by the LF_IP algorithm. The sixth column shows the improvement of the schedule length of the fused loop by the LF_IP algorithm. Table 2 displays the schedule length of the original loop and the fused loop when there are 8 function units in the DSP processor.

From Table 2, we find that the Basic_LF algorithm reduces the original schedule length by 54.5% on average when there are 8 FUs. While the LF_IP algorithm reduces the original schedule length by 68.9% on average when there are 8 FUs. It means that a better performance can be achieved by the LF_IP algorithm because the critical path of the fused loop by the LF_IP algorithm is shorter. The schedule length of the fused loop by the LF_IP algorithm achieves the minimal critical path length when there are enough function units. For example, the schedule length of the fused loop of LDG3 is minimal when there are 8 FUs as shown in Table 2.

5 Conclusion

In this paper, we presented an efficient loop fusion technique based on loop dependency graph model and multi-dimensional retiming concept. We derive legalizing fusion theorems to transform the loops to be legally fused. Polynomial-time legalizing fusion algorithms were developed to solve the loop fusion problem for both 1-level and 2-level loops. The experimental results showed that our loop fusion technique always significantly reduced the schedule length because we maximized the utilization of the Functional Units by exploiting more instruction level parallelism. Our future work will combine loop fusion with other loop transform techniques such as loop distribution and loop unrolling to further optimize the execution of the loops.

References

- [1] A. Darte. On the complexity of loop fusion. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 149–157, Oct. 1999.
- [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 *Proceedings 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, Springer-Verlag, Berlin*, pages 301–320, 1993.
- [4] K. Kennedy and K. S. McKinley. Typed fusion with applications to parallel and sequential code generation. Technical Report CRPC-TR94646, Center for Research on Parallel Computation, Rice University, Jan. 1994.
- [5] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, June 1991.
- [6] N. Manjikian and T. S. Abdelrahman. Fusion of loops for parallelism and locality. *IEEE Transactions on Parallel and Distributed System*, 8:193–209, Feb. 1997.
- [7] N. Megiddo and V. Sarkar. Optimal weighted loop fusion for parallel programs. In *Proceedings of the ninth annual ACM Symposium on Parallel Algorithms and Architectures*, pages 282–291, 1997.
- [8] T. W. O’Neil and E. H.-M. Sha. Minimizing inter-iteration dependencies for loop pipelining. In *ISCA 13th International Conference on Parallel and Distributed Computing Systems, Las Vegas, Nevada*, pages 412–417, Aug. 2000.
- [9] N. L. Passos and E. H.-M. Sha. Full parallelism of uniform nested loops by multi-dimensional retiming. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, volume 2, pages 130–133, 1994.
- [10] 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.
- [11] Y. Qian, S. Carr, and P. Sweany. Loop fusion for clustered VLIW architecture. In *Proceedings of the joint conference on Languages, compilers and tools for embedded systems: software and compilers for embedded systems*, pages 112–119, 2002.
- [12] E. H.-M. Sha, T. W. O’Neil, and N. L. Passos. Efficient polynomial-time nested loop fusion with full parallelism. *International Journal of Computers and Their Applications*, 10(1):9–24, Mar. 2003.
- [13] S. K. Singhai and K. S. McKinley. A parameterized loop fusion algorithm for improving parallelism and cache locality. *The Computer Journal*, 40(6):340–355, June 1997.
- [14] S. Verdoolaege, M. Bruynooghe, and F. Catthoor. Multi-dimensional incremental loop fusion for data locality. In *Proceedings of the Application-Specific Systems, Architectures, and Processors*, pages 14–24, 2003.
- [15] M. Wolfe. *High Performance Compilers for Parallel Computing*. Addison-Wesley Publishing Company, Inc., 1996.
- [16] 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.