Fusing Loops With Backward Inter Loop Data Dependence

Amit Ganesh
Computer Science Department
Stanford University,
Stanford, CA 94305
email: *amitg@cs.stanford.edu*

1 Introduction

Loop fusion has been a traditional transformation in optimizing and parallelizing compilers. There are various benefits in performing loop fusion. Most trivially, loop fusion reduces the overheads associated in implementing loops in computer hardware by a factor of 2. But, in addition, loop fusion also holds promise in reducing the paging traffic in the memory hierarchy [Kuck 84]. Moreover, loop fusion may also reduce register loads/stores and the overall memory requirements in certain cases [Wolfe 89].

This paper develops a general framework for fusing any two loops. We find the existing test for the legality of loop fusion to be unduly restrictive. Here, we show how to fuse loops which do not satisfy the condition in the data dependence test. Then, we extend the scope of performing loop fusion to adjacent loops with any inter loop dependence and also any nesting level. The tradeoff between the advantages of loop fusion and the disadvantages of loop unrolling would dictate whether to perform loop fusion on adjacent loops or not.

2 The Tests For Loop Fusion

There are two condition need to be met for fusing two adjacent loops [Wolfe 89, Zima 91]. Firstly, they should have identical loop limits. In case the loop limits differ by a small amount, loops may be partially unrolled and then fused.

Next, the traditional dependence test is applied. The dependence test forbids any dependence of the type $S_1\ \delta^*_>\ S_2$ where S1 and S2 are statements in the two loops. For example the following two loops are said to be non fusible:

```
        do I = 1, N
S1:        A[I] = B[I] + C[I]
        enddo

        do I = 1,N
S2:        C[I] = A[I+1] * 2
        enddo
```

The loops are not fusible because of the data dependence $S1\ \delta_>\ S2$. The iteration spaces for the loops is shown in figure 1. The dependence edges impose the order on the execution of the iterations in the corresponding loops. Direct fusion would result in the execution of $S2[1]$ before $S1[2]$ (i.e. the head of the

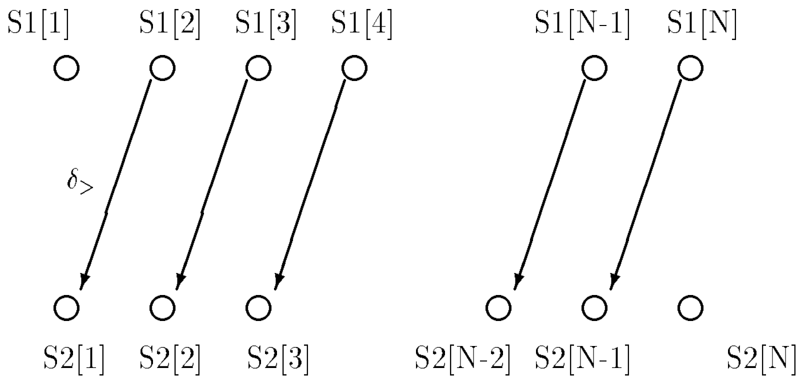

Figure 1: The data dependence between S1 and S2 (distance = +1)

edge is executed before the tail). Thus this would alter the semantics of the program. But we observe that unrolling the first loop by one iteration would make the loops directly fusible because no backward dependence would remain. Loop unrolling would result in the following situation:

```
        A[1] = B[1] + C[1]
        do I = 2, N
S1:        A[I] = B[I] + C[I]
        enddo
        do I = 2,N
S2:        C[I-1] = A[I] * 2
        enddo
        C[N] = A[N+1] * 2
```

Now the dependence relation is $S1\ \delta_=\ S2$. Thus, the loops may be fused directly resulting in:

```
        A[1] = B[1] + C[1]
        do I = 2, N
S1:        A[I] = B[I] + C[I]
S2:        C[I-1] = A[I] * 2
        enddo
        C[N] = A[N+1] * 2
```

This example illustrates that the dependence test imposes an undue restriction on loop fusion. Thus, loop fusion may be performed on any two adjacent loops with identical loop limits. In the next section we illustrate the general transformation for loop fusion.

3 Advanced Loop Fusion

Let us define the dependence $S_1\ \delta^*_{[v_1,v_2,\ldots,v_d]}\ S_2$ to mean that $\exists[i_1,i_2,\ldots,i_d]$ and $[j_1,j_2,\ldots,j_d]$ such that $S_1[i_1,i_2,\ldots,i_d]\ \delta^*\ S_2[j_1,j_2,\ldots,j_d]$ and $i_k - j_k = v_k\ \forall\ k = 1,d$. Thus we replace the notion of direction vectors by the more accurate dependence distance vectors. Positive, zero and negative dependence distances correspond to $>,=,<$ directions respectively.

The Fusion Transformation

Consider the two loops L1 and L2 with statements $S1$ and $S2$ respectively. Let a dependence $S1\ \delta_v\ S2$ exist. Let v_max be the largest such v. Assume that the loop limits are identical (this can be achieved by appropriately unrolling the loops). If v_max ≤ 0 then the loops can be fused as it is. If v_max > 0 (corresponding to the $\delta_>$ in the dependence test) then the following transformation is applied:

```
do I = 1, N
   S1[I]
enddo
do I = 1, N
   S2[I]
enddo
```

is transformed to

```
S1[1]
...                // the unrolled part of the first loop
S1[v_max]
do I = v_max + 1, N
   S1[I]
   S2[I-v_max]     // the new loop after fusion
enddo
S2[N-v_max+1]
...                // the unrolled part of the second loop
S2[N]
```

The transformation unrolls the first loop through v_max iterations from the lower limit and the second loops last v_max iterations. Then it schedules the remaining N $-$ v_max iterations of the two loops along the dependence edges representing $\delta^*_{\text{v\_max}}$. The legality of the transformation follows from lemma 1. Our definition of legality is different from the one used in [Wolf 91] for describing the transformations on single nested loops. A legal transformation would be one which would preserve all the dependences in the original program.

Lemma 1 *If there exists a data dependence, $S1\ \delta^*_v\ S2$ between the statements in the two adjacent loops of the original program, then the transformed program preserves this dependence.*

Proof : Consider any data dependence $S1\ \delta^*_v\ S2$ between the two loops. This implies that $\exists(i, j)$ such that the dependence is $S1[i]\ \delta^*\ S2[j]$ and $i - j = v$ in the original program.

In the transformed program, we observe that $S1[i]\ \delta^c\ S2[k]$ $\forall$ k $\geq$ i $-$ v_max. But, $j = i - v$ and $v \leq$v_max. That implies $j \geq i-$ v_max. Therefore, $S1[i]\ \delta^c\ S2[j]$. Thus, the data dependence $S1\ \delta^*_v\ S2$ is preserved by the transformation. □

This transformation suffers from the disadvantages of loop unrolling. In case v_max is large, the unrolled part of the loops can be rolled back into new loops thus saving on code space. This would incur an additional overhead of the loop tests, but is still at least as efficient as the original loops.

In the worst case, v_max may be equal to N $-$ 1 as in the following example:

```
        do I = 1, N
S1:        A[N-I+1] = B[I] * 3
        enddo
```

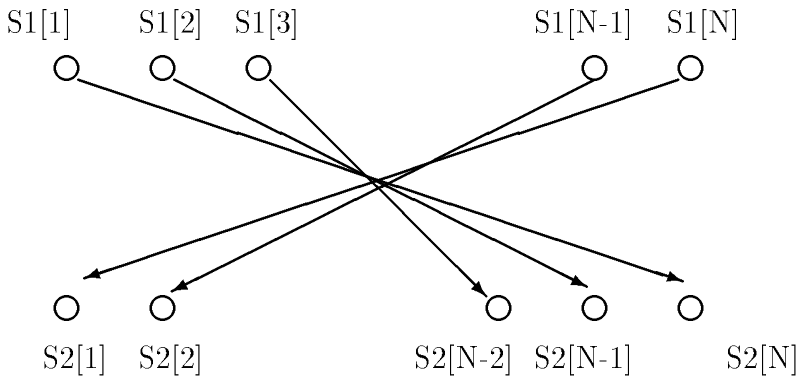

Figure 2: An example where direct fusion is of no use

```
        do I = 1, N
S2:        C[I] = A[I] + D[I]
        enddo
```

The iteration space diagram for these two loops is shown in figure 2. In this case, v_max is of the order of the range of the loop index. Here, both the loops need to be unrolled completely and loop fusion takes place on loops which consist of only one iteration (that of I = N). Such a loop fusion is not of any advantage. Therefore, the benefits of loop fusion using this transformation can be capitalized only when v_max is a small number. Interestingly, in some cases loop reversal may open new opportunities for loop fusion. Note that the condition for loop reversal, the absence of cross iteration dependencies, must be satisfied. For example, in the above example reversing the first loop by the substitution I = N − I + 1, makes the two loops directly fusible (since the dependence becomes $S1\ \delta_{=}\ S2$). This reversal of the first loop is equivalent to reflecting the line in its iteration space (refer figure 2) about its axis of symmetry. That leaves no backward dependence between the two loops.

Loop Fusion of Tightly Nested Loops

This transformation can be generalized to allow for fusion of tightly nested loops. Consider two statements S_1 and S_2 in two tightly nested loops L1 and L2. Consider a dependence $S_1\ \delta_{[v_1,v_2,\ldots,v_k,\ldots,v_d]}\ S_2$ where v_k is the largest across all the dependences between S_1 and S_2. Define v_k^{max} to be v_k if $v_k > 0$ and 0 otherwise. Now consider the dependence distance vector $v^{max} = [v_1^{max}, v_2^{max}, \ldots, v_d^{max}]$. This vector represents the amount of loop unrolling that needs to be applied to the two nested loops L1 and L2.

Consider two adjacent tightly nested loops:

```
do I1 = 1, N1
     ... do Id = 1, Nd       // nesting depth = d
         S1[I1, I2, ..., Id]
     ... enddo
enddo
do I1 = 1, N1
     ... do Id = 1, Nd       // nesting depth = d
          S2[I1, I2, ..., Id]
     ... enddo
enddo
```

The loops above will be fused as given below. It is easily seen that all the data dependencies will be satisfied by the fused loops.

```
/* unrolled part of S1[] here */
     do I1 = v_max_1+1, N
          ... do Id = v_max_d+1, N
               S1[I1, I2, ..., Id]
               S2[I1-v_max_1,
                  ...,
                  Id-v_max_d]
          ... enddo
     enddo
/* unrolled part of S2[] here */
```

The above transformation is a straightforward extension of the case of fusing two single iteration loops. The transformation can be visualized as translation along different directions followed by the merging of a n-dimensional plane (in the iteration space of the first loop nest) with another corresponding to the nested loops L1 and L2. For adjacent loops with nesting depth, d = 1, the 1-dimensional planes are lines and the dependencies are as seen in figure 1. We now prove the legality of the given transformation on lines similar to the proof of the transformation for ordinary (i.e. not nested) adjacent loops.

Lemma 2 *If there exists a data dependence, $S1\ \delta^*_{[v1,\ldots,vd]}\ S2$ between the statements in the two nested loops of the original program, then the transformed program preserves this dependence.*

Proof : Consider any data dependence $S1\ \delta^*_{[v1,\ldots,vd]}\ S2$ between the two loops. This implies that $\exists([i_1, i_2, \ldots, i_d], [j_1, j_2, \ldots, j_d])$ such that the dependence is $S1[i_1, \ldots, i_d]\ \delta^*\ S2[j_1, \ldots, j_d]$ and $i_l - j_l = v_l$ $\forall\ l = 1, d$ in the original program.

In the transformed program, we observe the dependence $S1[i_1, \ldots, i_d]\ \delta^c\ S2[k_1, \ldots, k_d]\ \forall\ k_l \geq i - v_l^{max}$ and $\forall\ l = 1, d$. But, $j_l = i_l - v_l$ and $v_l \leq v_l^{max}$. That implies $j_l \geq i_l - v_l^{max}$. Therefore, $S1[i_1, \ldots, i_d]\ \delta^c\ S2[j_1, \ldots, j_2]$. Thus, the data dependence $S1\ \delta^*_{[v_1,\ldots,v_d]}\ S2$ is preserved by the transformation. □

Due to the product of the dependence distances in different dimensions fusion of nested loops may lead to a large amount of unrolling of loops and thus may be disadvantageous. The amount of unrolling required may be much more than that in case of ordinary loops.

4 Further extensions

This framework for loop fusion may be extended to include *imperfectly* nested adjacent loops. The dependence distance of all the statements of the loops need to be considered. The evaluation of v_max_i remains the same: take the maximum dependence distance between any two statements (in separate loops) along the i-th axis in the iteration space.

Our transformation may result in a unrolled portion from both the original loops resulting in statements preceding and following the fused loop. These statements disallow the fusion of the new loop with other loops. The scope of loops fusion may be further extended by relaxing the condition of *adjacency* of the two loops. As long as the intervening statements are not data dependent on the loop above them, they can be hoisted above that loop and then the two loops can be transformed using our transformation. Otherwise, if the intervening statements have a data dependence with statements in the loop above, a part of that loop may be unrolled to enable the hoisting.

5 Conclusion

We observe that the traditional dependence test for the feasibility of loop fusion may be done away with. Instead of declaring loop fusion illegal for the loops failing the data dependence test, our transformations may be applied to fuse the loops. We extend the notion of loop fusion to nested loops and show how any two nested loops may be fused. We point out that loop fusion may also be applied to loops that are not tightly nested and loops which are not adjacent. We also note that in case of a large backward dependence distance between the loops, excessive unrolling may result and loop fusion may not be advantageous.

References

[Kuck 84] D.J. Kuck, R.H. Kuhn, B.R. Leasure. and M.J. Wolfe. The structure of an advanced retargetable vectorizer. In *Supercomputers: Design and Applications Tutorial*, pp 967-74, 1984.

[Wolf 91] M.E. Wolf, M.S. Lam. A Loop Transformation Theory and An Algorithm to Maximize Parallelism.*IEEE Transactions on Parallel and Distributed Systems*, Oct. 1991, pp452-471.

[Wolfe 89] Michael Wolfe. *Optimizing Supercompiler for Supercomputers.* MIT Press, Cambridge, Massachusetts, 1989.

[Zima 91] Hans Zima, Barbara Chapman. *Supercompilers for Parallel and Vector Computers*, ACM Press, New York, Addison-Wesley Publishing Company, 1991.