Removing Impediments to Loop Fusion Through Code Transformations

Bob Blainey, Christopher Barton, José Nelson Amaral
Presented by Mabel Chan, Kevin Wang, James Wu (Group 26)

What is loop fusion?

- Optimization technique that takes several loops and combines them into a single large loop
- Decreases the number of loop branches executed
- Creates opportunities for data reuse
- Generally we:
  - Scan the code to find pairs of normalized loops that can be fused
  - Greedily fuse them

Requirements for loop fusion

- Are control equivalent
- Have no dependencies
- Have conforming bounds *
- No intervening code between the loops *

*: these conditions are loosened in the proposed algorithm

Negative Effects of loop fusion

- Increased code size
- Increased register pressure within a loop
- Potential over-committing of hardware resources
- Formation of loops with more complex control flow

Loop Fusion Basic Example

```
for (int i = 0; i < 300; i++) {
   a[i] = a[i] + 3;
}

for (int i = 0; i < 300; i++) {
   b[i] = b[i] + 4;
}
```

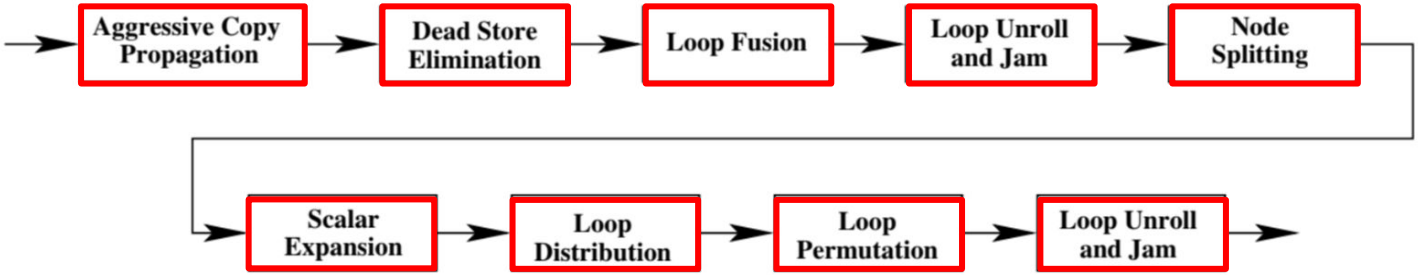
```
for (int i = 0; i < 300; i++) {
   a[i] = a[i] + 3;
   b[i] = b[i] + 4;
}
```

Problem: Loop fusion takes place under very specific conditions

Overview

- Code often has only small sets of loops that meet general code fusion requirements
- Lays out new algorithms to make code more fuseable
  - Eliminating conditions that prevent loop fusion
  - Generating maximal fusion -> increases scope of later transformations
- Focuses on IBM XL compiler suite
- Significant improvements to number of fused loops as a result of new algorithms

Conventional Loop Optimizations



- Optimization transformations in conventional XL compilers
- Paper's innovations focuses on loop fusion
  - Increases scope for optimization in the loop distributor

```
LoopFusion
1.  foreach NestLevel N_i from outermost to innermost
2.      Gather identically control dependent loops in N_i
        into LoopSets
3.      foreach LoopSet S_i
4.          Remove loops non-eligible for fusion from
            S_i
5.          FusedLoops ← True
6.          Direction ← Forward
7.          while FusedLoops = True
8.              if |S_i| < 2
9.                  break
10.             endif
11.             Build Control Flow Graph
12.             Compute Dominance Relation
13.             FusedLoops =
                    LoopFusionPass(S_i,Direction)
14.             if Direction = Forward
15.                 Direction = Reverse
16.             else
17.                 Direction = Forward
18.             endif
19.         endfor
20.     end while
21. endfor
```

Loop Fusion Algorithm

- New loop fusion algorithm is the main innovation of the paper
- Creates opportunities for loop fusion:
  - Can fuse loops that don't conform
  - Can fuse loops with intervening code
- Note: loops are normalized

Loop Fusion Pass Example

```
do i1 = 1, n
   a(i1) = a(i1) * k1
end do
do i2 = 1, n-1
   d(i2) = a(i2) - b(i2+1) * k2
end do
ds  = 0.0
do i3 = 1, m
  ds = ds + d(i3)
end do
if(n<m)
  c(n-2) = n
else
  c(n-2) = m
do i4 = 1, n-2
  b(i4) = a(i4) + b(i4) / c(i4)
end do
```

(b) Fortran 77 Code

```
do i5 = 1, n
   if (i5 < n-1)
      a(i5) = a(i5) * k1
      d(i5) = a(i5) - b(i5+1) * k2
   else
      a(i5) = a(i5) * k1
end do
ds  = 0.0
do i3 = 1, m
  ds = ds + d(i3)
end do
if(n<m)
  c(n-2) = n
else
  c(n-2) = m
do i4 = 1, n-2
  b(i4) = a(i4) + b(i4) / c(i4)
end do
```

(a) After Fusing i1 and i2 into i5

```
ds  = 0.0
if(n<m)
  c(n-2) = n
else
  c(n-2) = m
do i5 = 1, n
   if (i5 < n-1)
      a(i5) = a(i5) * k1
      d(i5) = a(i5) - b(i5+1) * k2
   else
      a(i5) = a(i5) * k1
end do
do i3 = 1, m
  ds = ds + d(i3)
end do
do i4 = 1, n-2
  b(i4) = a(i4) + b(i4) / c(i4)
end do
```

(b) After moving intervening code up

Fig. 6. Completing first forward pass in running example

```
LoopFusion
1.  foreach NestLevel N_i from outermost to innermost
2.          Gather identically control dependent loops in N_i
            into LoopSets
3.          foreach LoopSet S_i
4.                  Remove loops non-eligible for fusion from
                    S_i
5.                  FusedLoops ← True
6.                  Direction ← Forward
7.                  while FusedLoops = True
8.                          if |S_i| < 2
9.                            break
10.                         endif
11.                         Build Control Flow Graph
12.                         Compute Dominance Relation
13.                         FusedLoops =
                              LoopFusionPass(S_i,Direction)
14.                         if Direction = Forward
15.                           Direction = Reverse
16.                         else
17.                           Direction = Forward
18.                         endif
19.                 endfor
20.         end while
21. endfor
```

Loop Fusion Algorithm

- Loop through each nest level:
- Partitions all loops into sets of loops that are control equivalent
- Loop through each loop set:
- Remove loops non-eligible for fusion from the set

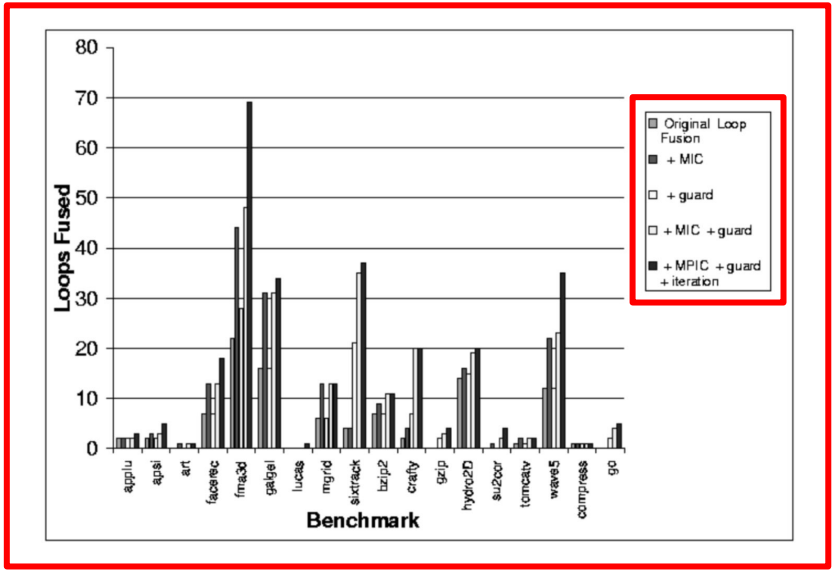
- Alternate forward and reverse passes over the set to fuse loops until no more loops can be fused
- CFG and dominance relations recomputed in each iteration

```
LoopFusionPass(S_i, Direction)
1.  FusedLoops = False
2.  foreach pair of loops L_j and L_k in S_i, such that L_j
        dominates L_k, in Direction
3.      if InterveningCode(L_j, L_k) = True and
           IsInterveningCodeMovable(L_j, L_k) = False
4.          continue
5.      endif
6.      σ ← |κ(L_j) − κ(L_k)|
7.      if L_j and L_k are non-conforming and
           σ cannot be determined at compile time
8.          continue
9.      endif
10.     if DependenceDistance(L_j, L_k) < 0
11.         continue
12.     endif
13.     MoveInterveningCode(L_j, L_k, Direction)
14.     if InterveningCode(L_j, L_k) = False
15.         if L_j and L_k are non-conforming
16.             L_m ← FuseWithGuard(L_j, L_k)
17.         else
18.             L_m ← Fuse(L_j, L_k)
19.         endif
20.         S_i ← S_i ∪ L_m − {L_j, L_k}
21.         FusedLoops = True
22.     else
23.         continue
24.     endif
25. endfor
26. return FusedLoops
```

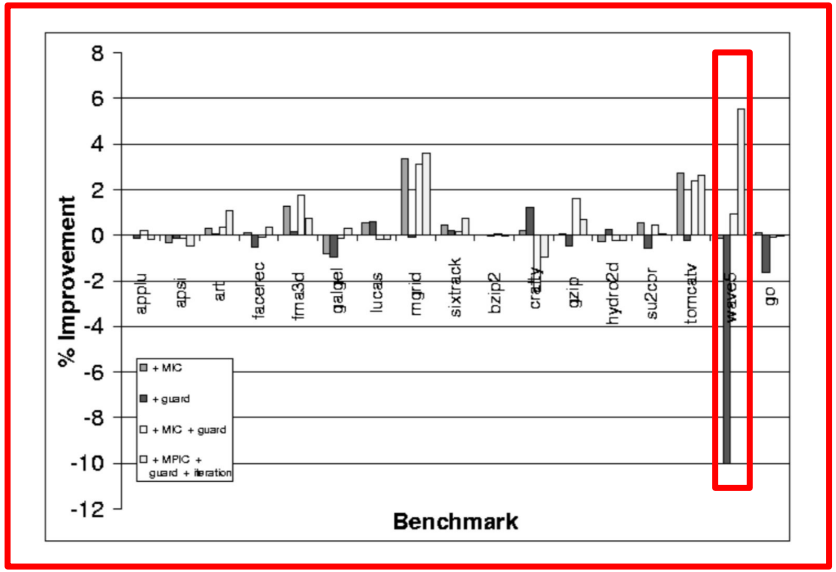
Loop Fusion Algorithm

- Iterate through loop pairs in dominance order if forward, etc
- If the loops aren't adjacent and intervening code can't be moved; or the loops don't conform and the difference btwn. their upper bounds can't be determined; or there's neg. dependence distance between the loops; can't fuse
- Move intervening code
- Fuse conforming loops, fuse non-conforming loops with guard

Results



Number of loops fused with each version of the compiler



Execution times for selected benchmarks

Conclusion

- Strengths
  - Increased number of loops fused
  - Enables technology for optimization to take place later in this compiler framework, ie, loop distribution
  - Increase granularity of parallelism and minimize loop synchronization
- Weaknesses
  - Inhibit software pipelining
- Future Works
  - Removing restrictions that all loops that are fused must be control equivalent
  - Variation of index set splitting to remove control flow splits
  - Loop Dependence Graph to solve data cache performance

Q&A