Revisiting Loop Fusion and its place in the loop transformation framework

October 18, 2018

Kit Barton, IBM Canada
Johannes Doerfert, Argonne National Labs
Hal Finkel, Argonne National Labs
Michael Kruse, Argonne National Labs

Agenda

Motivation and Goals

Loop representation in LLVM

Algorithm for loop fusion

Current Results

Next Steps

Loop Fusion

Combine two (or more) loops into a single loop

```
for (int i=0; i < N; ++i) {
  A[i] = i;
}
for (int j=0; j < N; ++j) {
  B[j] = j;
}
```

```
for (int i=0, j=0; i < N && j < N; ++i,++j) {
  A[i] = i;
  B[j] = j;
}
```

Motivation

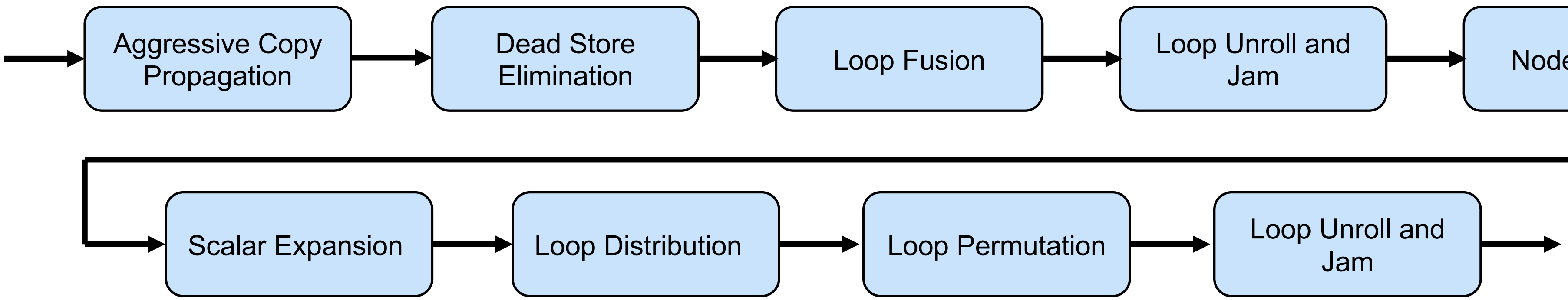
- Data reuse, parallelism, minimizing bandwidth, …
- Increase scope for loop optimizations

Our Goals

1. Way to learn how to implement a loop optimization in LLVM
2. Starting point for establishing a loop optimization pipeline in LLVM

XL Loop Optimization Pipeline

IBM's XL Compiler has a very mature loop optimization pipeline



The pipeline begins with **maximal fusion** – greedily fuse loops to create large loop nests

Run a series of loop optimizations on the loop nests created by fusion

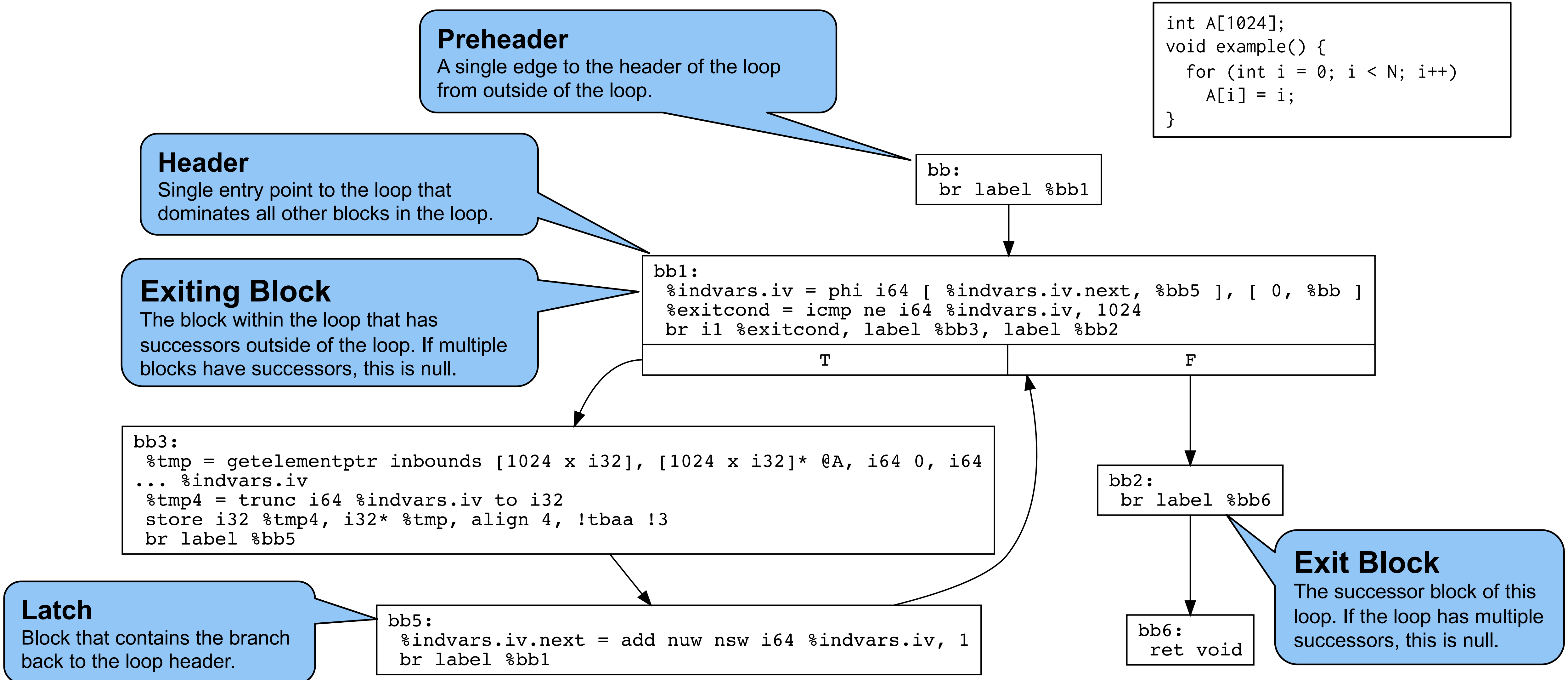
Selectively distribute loops based on a set of heuristics, including:

- data reuse
- independent loops
- perfect loop nests
- register pressure
- ...

Christopher Barton. *Code transformations to augment the scope of loop fusion in a production compiler*. Master's thesis, University of Alberta, January 2003.

Loop Representation in LLVM

```
int A[1024];
void example() {
  for (int i = 0; i < N; i++)
    A[i] = i;
}
```



CFG for 'example' function

Requirements for loop fusion

In order for two loops, L_j and L_k to be fused, they must satisfy the following conditions:

1. *L_j and L_k must be adjacent*

   There cannot be any statements that execute between the end of L_j and the beginning of L_k

2. *L_j and L_k must iterate the same number of times*

3. *L_j and L_k must be control flow equivalent*

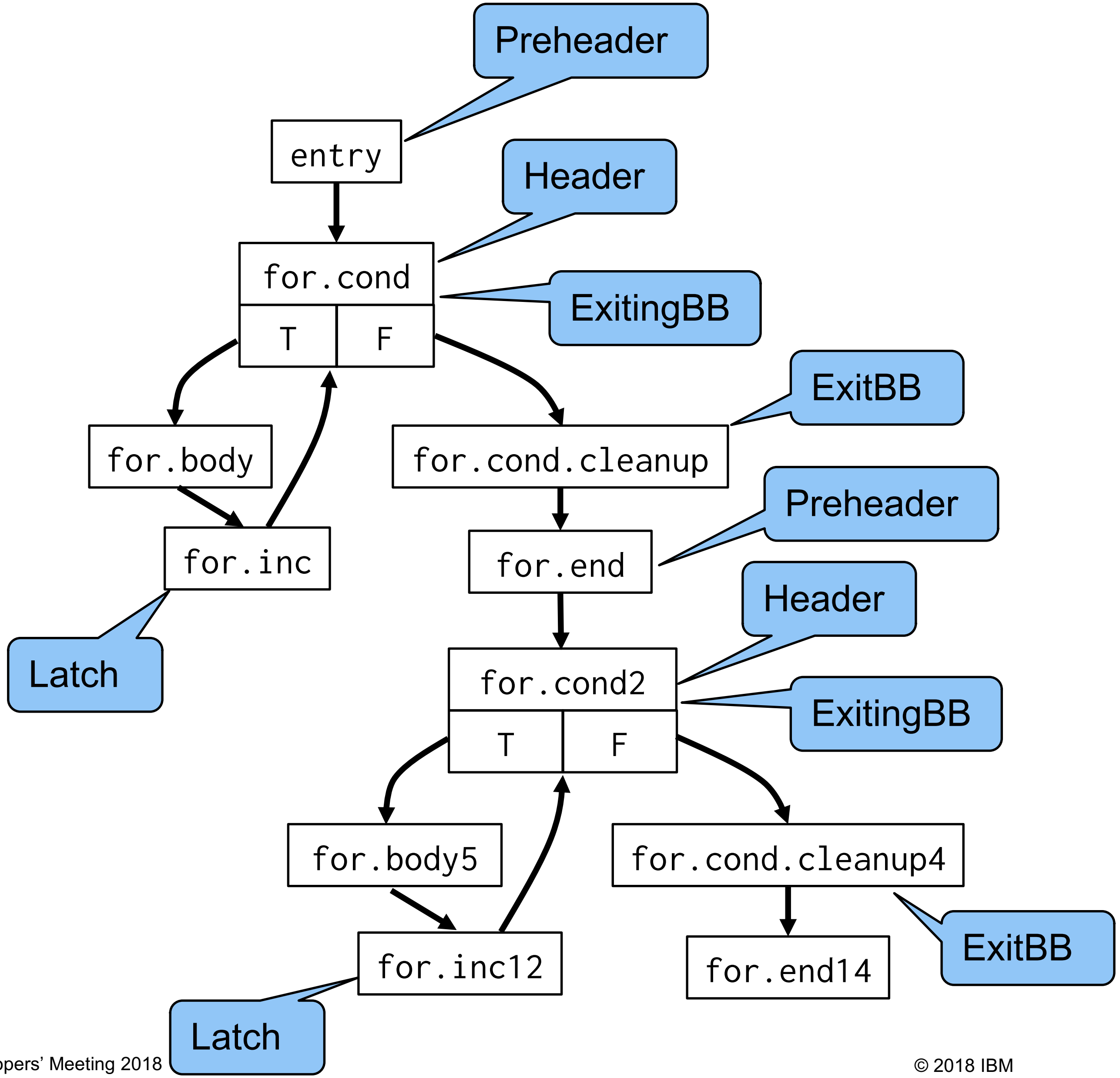
   When L_j executes L_k also executes or when L_k executes L_j also executes

4. *There cannot be any negative distance dependencies between L_j and L_k*

   A negative distance dependence occurs between L_j and L_k, L_j before L_k, when at iteration m L_k uses a value that is computed by L_j at a future iteration m+n (where n > 0).

Loop Fusion Algorithm

fuseLoops(Function F)
- for each nest level NL, outermost to innermost
  - Collect loops that are candidates for loop fusion at NL
  - Sort candidates into control-flow equivalent sets
  - for each CFE set
    - for each pair of loops, L_j and L_k
      - if L_j and L_k do not have identical trip counts
        - **continue**
      - if L_j and L_k cannot be made adjacent then
        - **continue**
      - if L_j and L_k have invalid dependencies then
        - **continue**
      - if fusing L_j and L_k is not beneficial then
        - **continue**
      - Move intervening code to make L_j and L_k adjacent
      - fuse L_j and L_k
      - Update fusion candidate list



Loop Fusion – collect candidates

fuseLoops(Function F)
  for each nest level NL, outermost to innermost
    Collect loops that are candidates for loop fusion at NL
    Sort candidates into control-flow equivalent sets
    for each CFE set
      for each pair of loops, L_j and L_k
        if L_j and L_k do not have identical trip counts
          continue
        if L_j and L_k cannot be made adjacent then
          continue
        if L_j and L_k have invalid dependencies then
          continue
        if fusing L_j and L_k is not beneficial then
          continue
        Move intervening code to make L_j and L_k adjacent
        fuse L_j and L_k
        Update fusion candidate list

Loops are not candidates for fusion if:

- They might throw an exception
- They contain volatile memory accesses
- They are not in simplified form
- Any of the necessary information is not available (preheader, header, latch, exiting blocks, exit block)

Loop Fusion – sort based on control-flow equivalence

fuseLoops(Function F)
  for each nest level NL, outermost to innermost
    Collect loops that are candidates for loop fusion at NL
    Sort candidates into control-flow equivalent sets
    for each CFE set
      for each pair of loops, L_j and L_k
        if L_j and L_k do not have identical trip counts
          continue
        if L_j and L_k cannot be made adjacent then
          continue
        if L_j and L_k have invalid dependencies then
          continue
        if fusing L_j and L_k is not beneficial then
          continue
        Move intervening code to make L_j and L_k adjacent
        fuse L_j and L_k
        Update fusion candidate list

Dominator and post-dominator trees are used to determine control-flow equivalence:

- if L_j dominates L_k and L_k post-dominates L_j then L_j and L_k are control-flow equivalent

Build sets of candidates that are all control flow equivalent by comparing a new loop to the first loop in a set.

Once all loops have been placed into sets, sets with a single loop are discarded.

Remaining set(s) are sorted in dominance order:

- if L_j is located in the set before L_k, then L_j dominates L_k

Loop Fusion – check trip counts

fuseLoops(Function F)
  for each nest level NL, outermost to innermost
    Collect loops that are candidates for loop fusion at NL
    Sort candidates into control-flow equivalent sets
    for each CFE set
      for each pair of loops, L_j and L_k
        if L_j and L_k do not have identical trip counts
          continue
        if L_j and L_k cannot be made adjacent then
          continue
        if L_j and L_k have invalid dependencies then
          continue
        if fusing L_j and L_k is not beneficial then
          continue
        Move intervening code to make L_j and L_k adjacent
        fuse L_j and L_k
        Update fusion candidate list

Scalar Evolution (SCEV) is used to determine trip counts

- If it cannot compute trip counts, or determine that the trip counts are identical, loops are not fused

We currently do not try to make trip counts the same via peeling

- This needs to be added in the future to enable more loop optimizations
- Interaction with other loop optimizations will be critical here

Loop Fusion – check adjacent

fuseLoops(Function F)
  for each nest level NL, outermost to innermost
    Collect loops that are candidates for loop fusion at NL
    Sort candidates into control-flow equivalent sets
    for each CFE set
      for each pair of loops, L_j and L_k
        if L_j and L_k do not have identical trip counts
          continue
        if L_j and L_k cannot be made adjacent then
          continue
        if L_j and L_k have invalid dependencies then
          continue
        if fusing L_j and L_k is not beneficial then
          continue
        Move intervening code to make L_j and L_k adjacent
        fuse L_j and L_k
        Update fusion candidate list

Analyze all instructions between the exit of L_j and the preheader of L_k and determine if they can be move prior to L_j or past L_k

Build a map of all instructions and the location where they can move (prior, past, both, none)

If any instructions cannot be moved, the two loops cannot be made adjacent and thus cannot be fused

Loop Fusion – check dependencies

fuseLoops(Function F)
- for each nest level NL, outermost to innermost
  - Collect loops that are candidates for loop fusion at NL
  - Sort candidates into control-flow equivalent sets
  - for each CFE set
    - for each pair of loops, L_j and L_k
      - if L_j and L_k do not have identical trip counts
        continue
      - if L_j and L_k cannot be made adjacent then
        continue
      - if L_j and L_k have invalid dependencies then
        continue
      - if fusing L_j and L_k is not beneficial then
        continue
      - Move intervening code to make L_j and L_k adjacent
      - fuse L_j and L_k
      - Update fusion candidate list

Three different algorithms are used to test dependencies for fusion:

- Alias Analysis
  - Test if two memory locations alias each other
- Dependence Info
  - Uses the depends interface from Dependence Info
- SCEV
  - Use SCEV to determine if there could be negative dependencies between the two loops

If **any** can prove valid dependencies, then fusion is legal

Loop Fusion – profitability analysis

fuseLoops(Function F)
  for each nest level NL, outermost to innermost
    Collect loops that are candidates for loop fusion at NL
    Sort candidates into control-flow equivalent sets
    for each CFE set
      for each pair of loops, L_j and L_k
        if L_j and L_k do not have identical trip counts
          continue
        if L_j and L_k cannot be made adjacent then
          continue
        if L_j and L_k have invalid dependencies then
          continue
        if fusing L_j and L_k is not beneficial then
          continue
        Move intervening code to make L_j and L_k adjacent
        fuse L_j and L_k
        Update fusion candidate list

Profitability Analysis

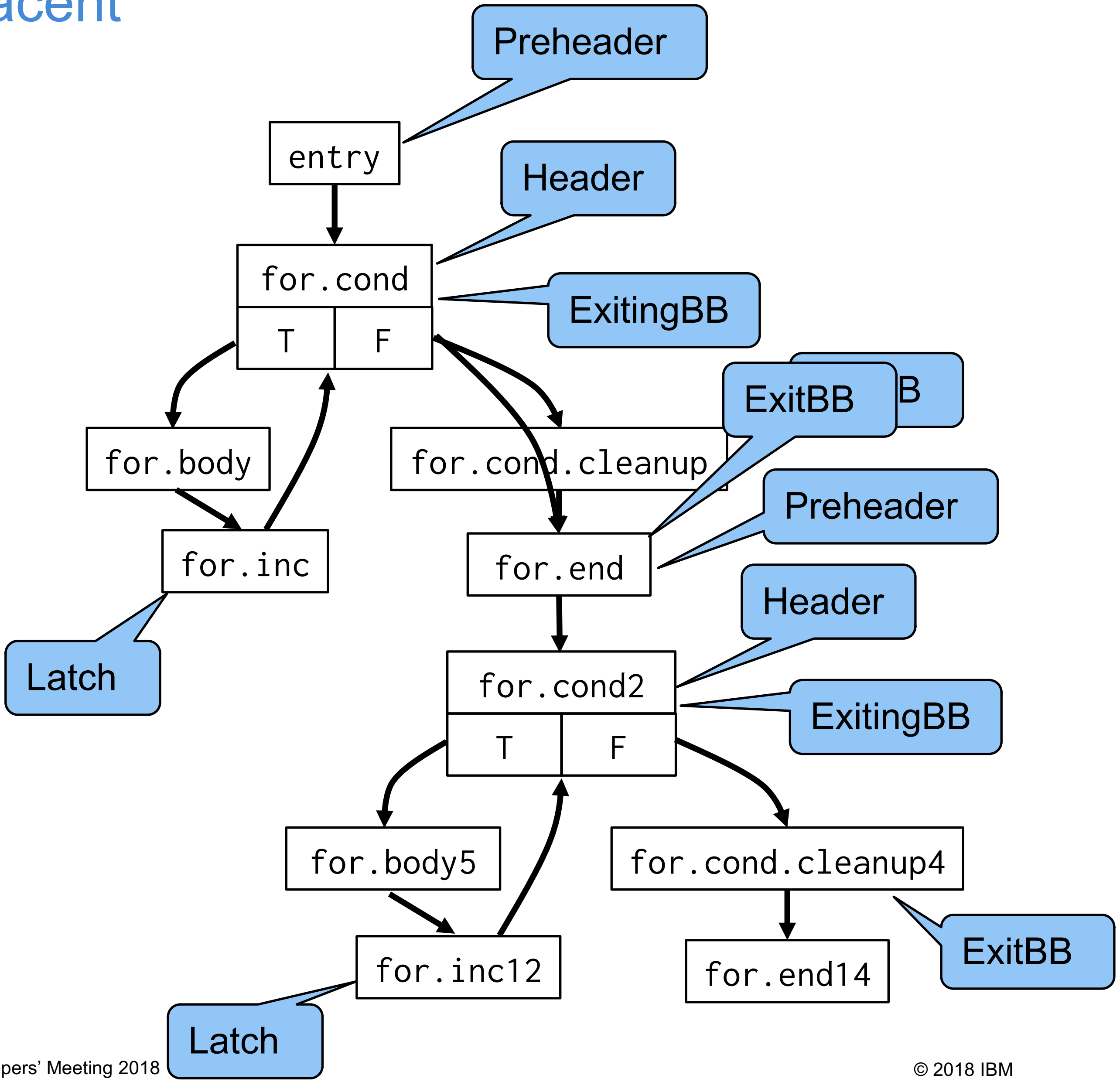
Hook that will allow different heuristics to be used to determine whether loops should be fused

Currently this always returns true, to allow maximal fusion

Loop Fusion – move code to make adjacent

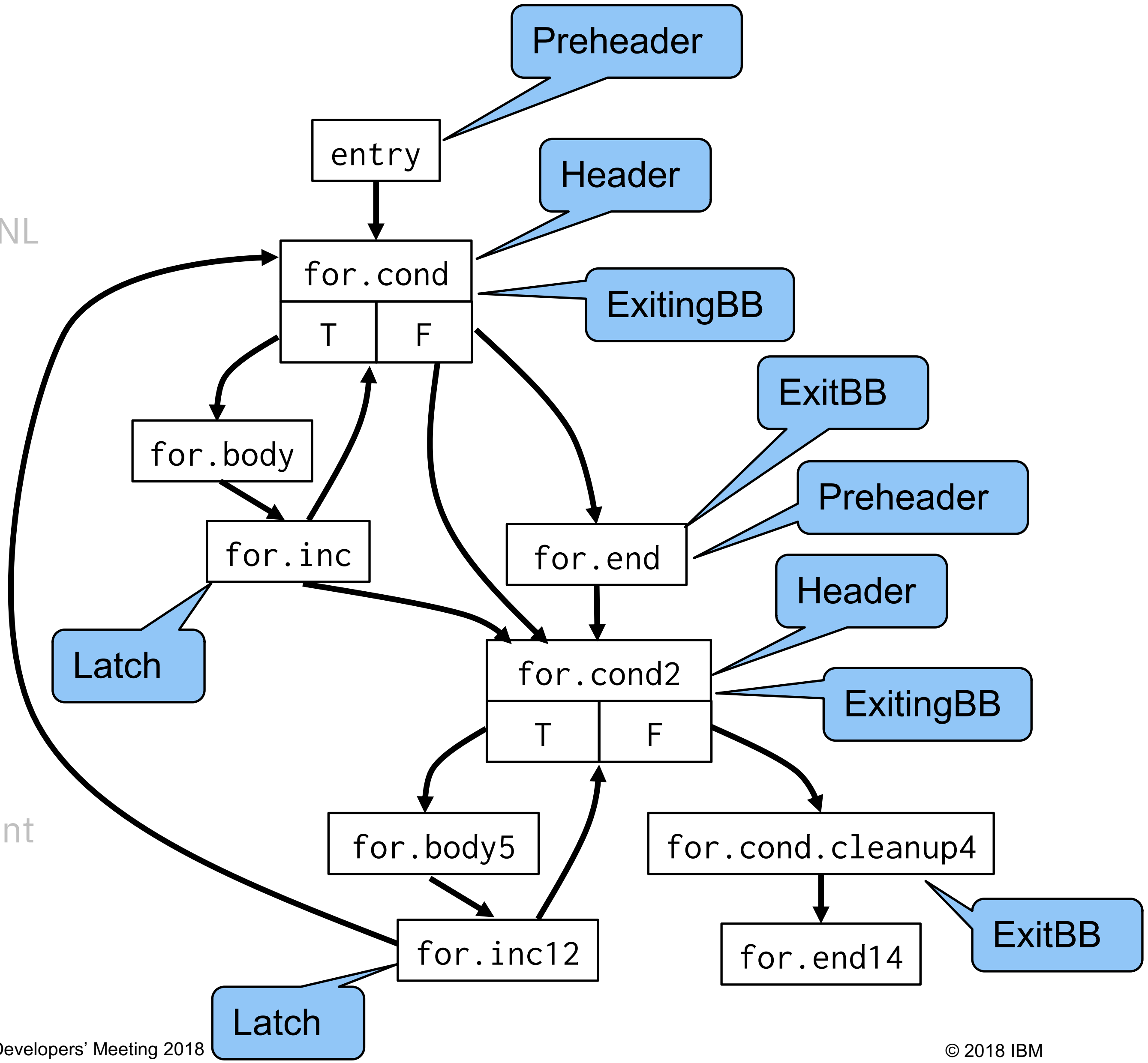
fuseLoops(Function F)

- for each nest level NL, outermost to innermost
  - Collect loops that are candidates for loop fusion at NL
  - Sort candidates into control-flow equivalent sets
  - for each CFE set
    - for each pair of loops, L_j and L_k
      - if L_j and L_k do not have identical trip counts
        continue
      - if L_j and L_k cannot be made adjacent then
        continue
      - if L_j and L_k have invalid dependencies then
        continue
      - if fusing L_j and L_k is not beneficial then
        continue
      - Move intervening code to make L_j and L_k adjacent
      - fuse L_j and L_k
      - Update fusion candidate list



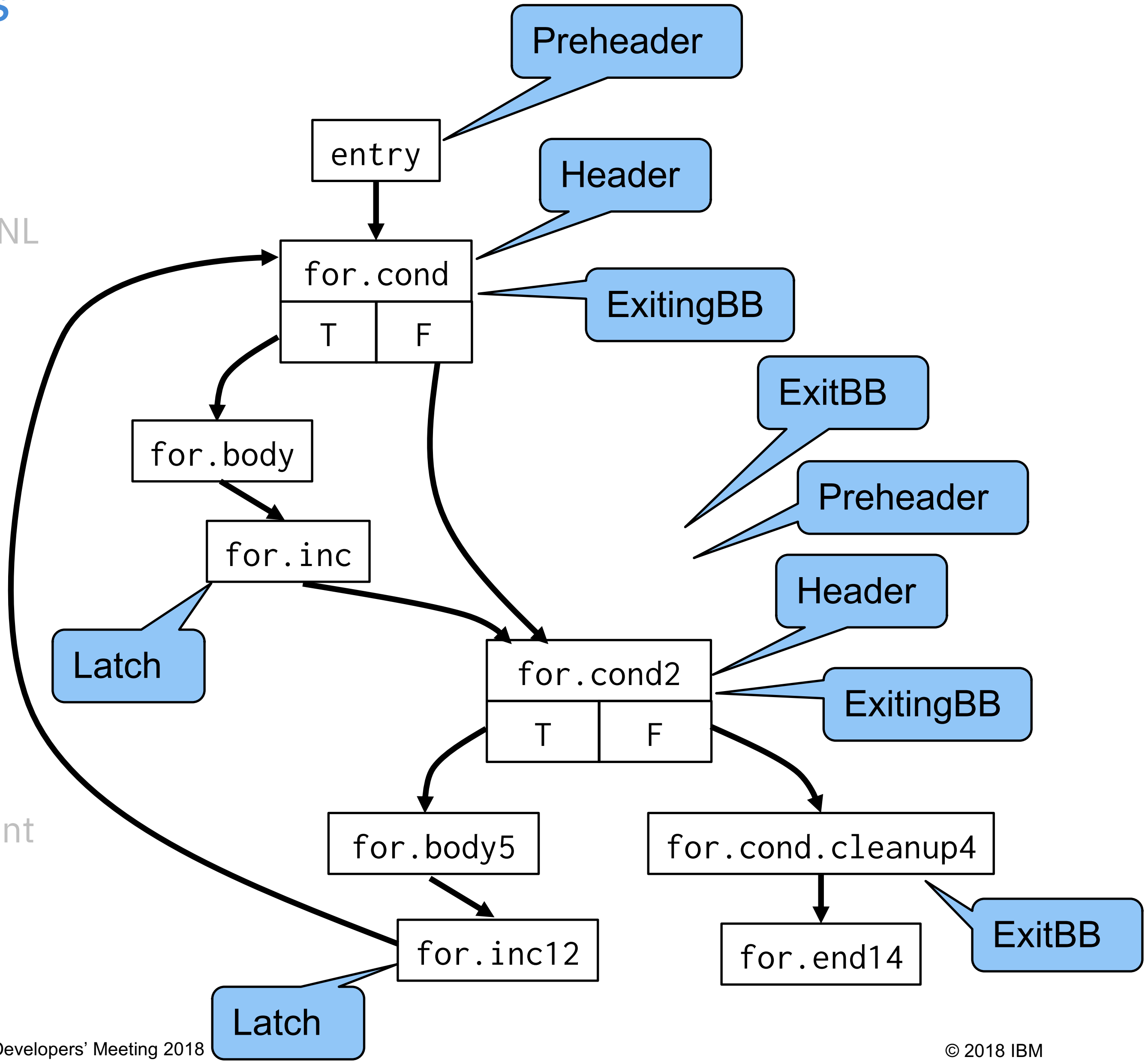
Loop Fusion – fuse loops

fuseLoops(Function F)
- for each nest level NL, outermost to innermost
  - Collect loops that are candidates for loop fusion at NL
  - Sort candidates into control-flow equivalent sets
  - for each CFE set
    - for each pair of loops, L_j and L_k
      - if L_j and L_k do not have identical trip counts
        - **continue**
      - if L_j and L_k cannot be made adjacent then
        - **continue**
      - if L_j and L_k have invalid dependencies then
        - **continue**
      - if fusing L_j and L_k is not beneficial then
        - **continue**
      - Move intervening code to make L_j and L_k adjacent
      - fuse L_j and L_k
      - Update fusion candidate list



Loop Fusion – update data structures

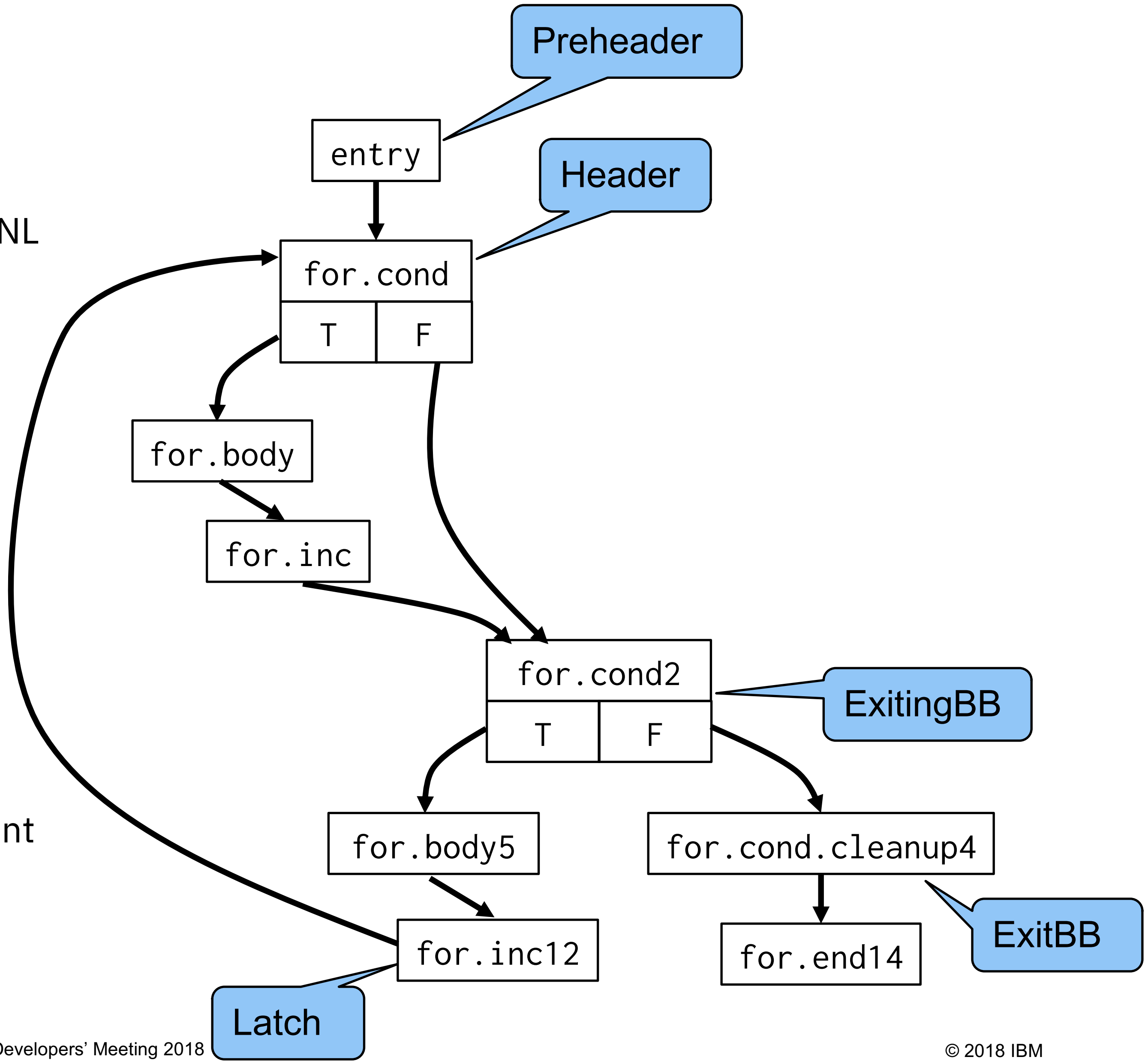
fuseLoops(Function F)
- for each nest level NL, outermost to innermost
  - Collect loops that are candidates for loop fusion at NL
  - Sort candidates into control-flow equivalent sets
  - for each CFE set
    - for each pair of loops, L_j and L_k
      - if L_j and L_k do not have identical trip counts
        continue
      - if L_j and L_k cannot be made adjacent then
        continue
      - if L_j and L_k have invalid dependencies then
        continue
      - if fusing L_j and L_k is not beneficial then
        continue
      - Move intervening code to make L_j and L_k adjacent
      - fuse L_j and L_k
      - Update fusion candidate list



After Loop Fusion

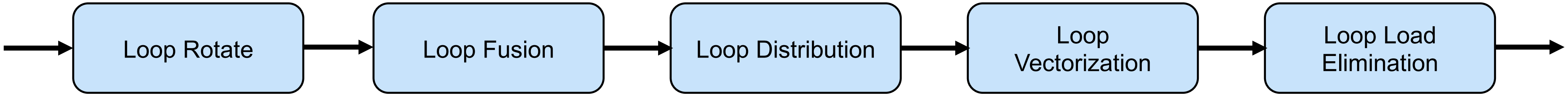
fuseLoops(Function F)

- for each nest level NL, outermost to innermost
  - Collect loops that are candidates for loop fusion at NL
  - Sort candidates into control-flow equivalent sets
  - for each CFE set
    - for each pair of loops, L_j and L_k
      - if L_j and L_k do not have identical trip counts
        continue
      - if L_j and L_k cannot be made adjacent then
        continue
      - if L_j and L_k have invalid dependencies then
        continue
      - if fusing L_j and L_k is not beneficial then
        continue
      - Move intervening code to make L_j and L_k adjacent
      - fuse L_j and L_k
      - Update fusion candidate list

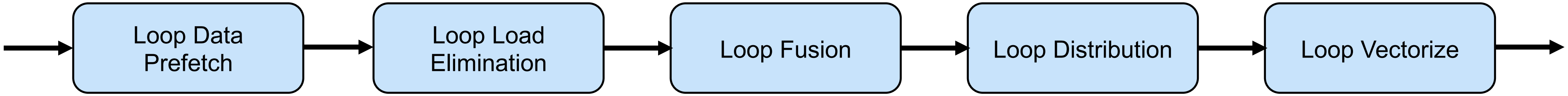


Current placement of Loop Fusion

Old Pass Manager



New Pass Manager



Number of Loops Fused

SPEC 2006

Benchmark	Candidates for Fusion	Loops Fused
perlbench	5	0
bzip2	21	1
gcc	50	8
namd	4	0
gobmk	96	7
dealII	355	0
soplex	1	0
povray	14	3
hmmer	19	0
h264ref	159	2
lbm	1	1
astar	7	0
sphinx3	8	1

SPEC 2017

Benchmark	Candidates for Fusion	Loops Fused
perlbench_r	7	0
gcc_r	114	2
namd_r	11	0
parest_r	137	1
povray_r	22	6
lbm_r	1	1
omnetpp_r	19	0
x264_r	81	6
blender_r	259	6
deepsjeng_r	34	0
imagick_r	45	5
nab_r	9	0
xz_r	18	1

Reasons for not fusing

SPEC 2006

Benchmark	Dependencies	Non-equal Trip Count	Cannot make adjacent
perlbench	1	2	2
bzip2	1	60	9
gcc	17	13	45
namd	0	3	3
gobmk	41	17	231
dealII	278	31	470
soplex	0	0	1
povray	0	17	4
hmmer	7	1	17
h264ref	105	74	506
astar	3	5	0
sphinx3	0	5	3

SPEC 2017

Benchmark	Dependencies	Non-equal Trip Count	Cannot make adjacent
perlbench_r	0	6	2
gcc_r	47	80	107
namd_r	8	1	3
parest_r	43	6	485
povray_r	3	36	7
omnetpp_r	0	29	0
x264_r	42	26	67
blender_r	164	53	310
deepsjeng_r	30	33	136
imagick_r	15	24	56
nab_r	0	12	7
xz_r	0	33	66

Ineligible Loops

SPEC 2006

Benchmark	May Throw Exception	Contains Volatile Access	Invalid Exiting Blocks	Invalid Exit Block	Invalid Trip Count
perlbench	0	62	451	485	340
bzip2	0	0	76	76	122
gcc	0	6	1643	1697	1418
mcf	0	0	8	8	11
milc	0	0	13	13	116
namd	4	0	91	91	67
gobmk	0	0	274	280	157
dealII	810	0	2098	2106	1319
soplex	98	0	155	155	98
povray	448	0	408	426	156
hmmer	0	0	156	157	246
libquantum	0	0	3	3	15
h264ref	0	0	80	81	225
omnetpp	70	0	174	183	80
astar	16	0	8	8	5
sphinx3	0	0	87	87	203
xalancbmk	704	0	1817	1888	1062

SPEC 2017

Benchmark	May Throw Exception	Contains Volatile Access	Invalid Exiting Blocks	Invalid Exit Block	Invalid Trip Count
perlbench_r	0	18	850	875	736
gcc_r	0	0	2864	2923	4012
mcf_r	0	0	15	15	26
namd_r	67	0	75	75	71
parest_r	545	0	2147	2147	3816
povray_r	435	0	421	439	167
omnetpp_r	293	0	400	415	321
xalancbmk_r	2477	0	2450	2526	1344
x264_r	0	0	211	212	500
blender_r	31	10	3058	3067	5952
deepsjeng_r	71	0	30	32	6
imagick_r	0	0	526	542	2387
leela_r	32	0	97	97	71
nab_r	0	0	117	163	323
xz_r	0	0	74	75	61

Next steps

Post patch

Investigate location to run loop fusion

Enhancements to fuse more

- Non-equal trip counts
  - Loop peeling or splitting
- Dependencies
  - Loop alignment or skewing