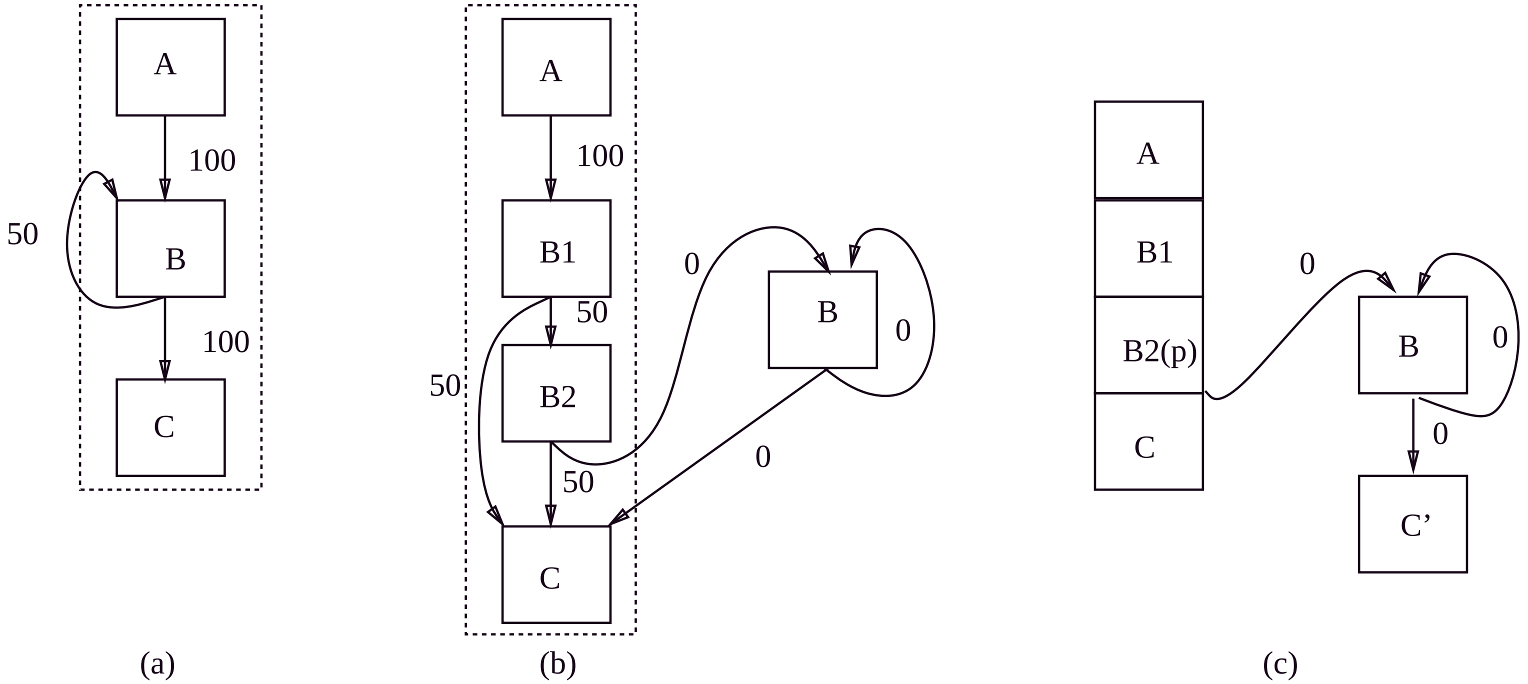
Hyperblock Peeling Optimizations

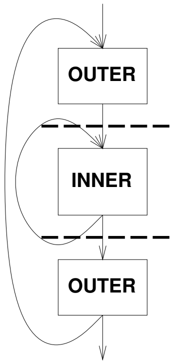
David I. August

Illinios IMPACT Compiler Project
University of Illinois at Urbana-Champaign
July 17, 1995

Review - What is Loop Peeling?



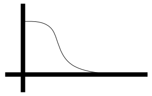
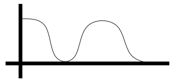
Example weighted control flow graph to illustrate hyperblock loop peeling, (a) original graph, (b) graph after loop peeling, (c) graph after loop peeling, tail duplication, and if-conversion.



Why is peeling important?

- There is a need to handle frequently executed paths uniformly regardless of loops.
  - Nested loops need to be handled more effectively.
    - Low iteration count - not enough iterations for effective overlap.
    - Very sequential - overlapping iterations is not effective.
  - Code surrounding nested loops needs to be handled more effectively.
    - No/Low iteration count - overlapping iterations ineffective.
    - Very sequential - other source of ILP is needed.
    - Nested loops create artificial barrier to code motion.
- Hyperblocks cannot have internal cycles.
  - Loops can be excluded from hyperblocks. This limits opportunities in hyperblock formation.
  - Alternatively, loops can be peeled and included in hyperblocks.

Where should we peel?

- Low iteration count inner loops: 
  - Example: Iter1-3[100], Iter4-∞[$\approx$0]
  - 3 or fewer iterations not enough to sustain desired ILP.
  - ILP of the program can be increased by including the first 3 iterations in the code surrounding the loop.
  - Peeling 3 times keeps execution in hyperblock trace. Program rarely leaves hyperblock to handle iterations 4-∞.
- Low iteration count with High iteration count: 
  - Example: Iter1-3[100], Iter4-49[$\approx$0], Iter50-100[100], Iter101-∞[$\approx$0]
  - Loop peeling the loop 3 times for inclusion in hyperblock avoids wasted work for low iteration counts.
  - A high iteration count version of the recovery loop can be optimized while safely ignoring low iteration count cases.
  - In effect, the loop is versioned.

Where else should we peel?

- Sparse surrounding region:
  - Peel enough times to fill empty slots in hyperblock.
  - Determining the number of peels is more difficult. Easy to do more harm than good.
    - Resource dependence overuse could result.
    - Dependence height of surrounding region could be increased.
- The inner loop and surrounding region relationships discussed can be applied to any depth of nested loops.
  - Entire loop nests can be peeled and share a common recovery loop nest.
  - Even more flexibility is given to hyperblock formation.

How many times do we peel?

- Peeling the right amount of times is critical.
  - Too few: recovery code entered too often and hyperblock is ineffective.
  - Too many: many useless instructions are fetched and dependence height increased.
- Current heuristic to identify candidates for peeling and determine how many times to peel:
  - What it does now:
    * There is a maximum number of ops a peeled loop code can have. This is used as an estimate of dependence height and resource usage.
    * The peeled loop must cover a certain amount of the loop invocations without going to the recovery loop.
    * Additional peels need to add a minimum amount of extra coverage to be included.
  - Doesn't take into account dependence height or available resources directly.
  - Doesn't identify sparse outer loop cases.

Running code example

```
#define DEPTH 3
#define FIND_LIST_SIZE 10

void access_hash_table()
{
  int list_indx, bucket_indx;
  for(list_indx = 0; list_indx < FIND_LIST_SIZE; list_indx++)
    {
      for(bucket_indx = 0; bucket_indx < DEPTH; bucket_indx++)
        {
          int element = find_list[list_indx];
          if(hash_table_element[element & 7][bucket_indx] == element)
            {
              hits++;
              break;
            }
        }
      if(bucket_indx == DEPTH)
        {
          misses++;
        }
    }
}
```

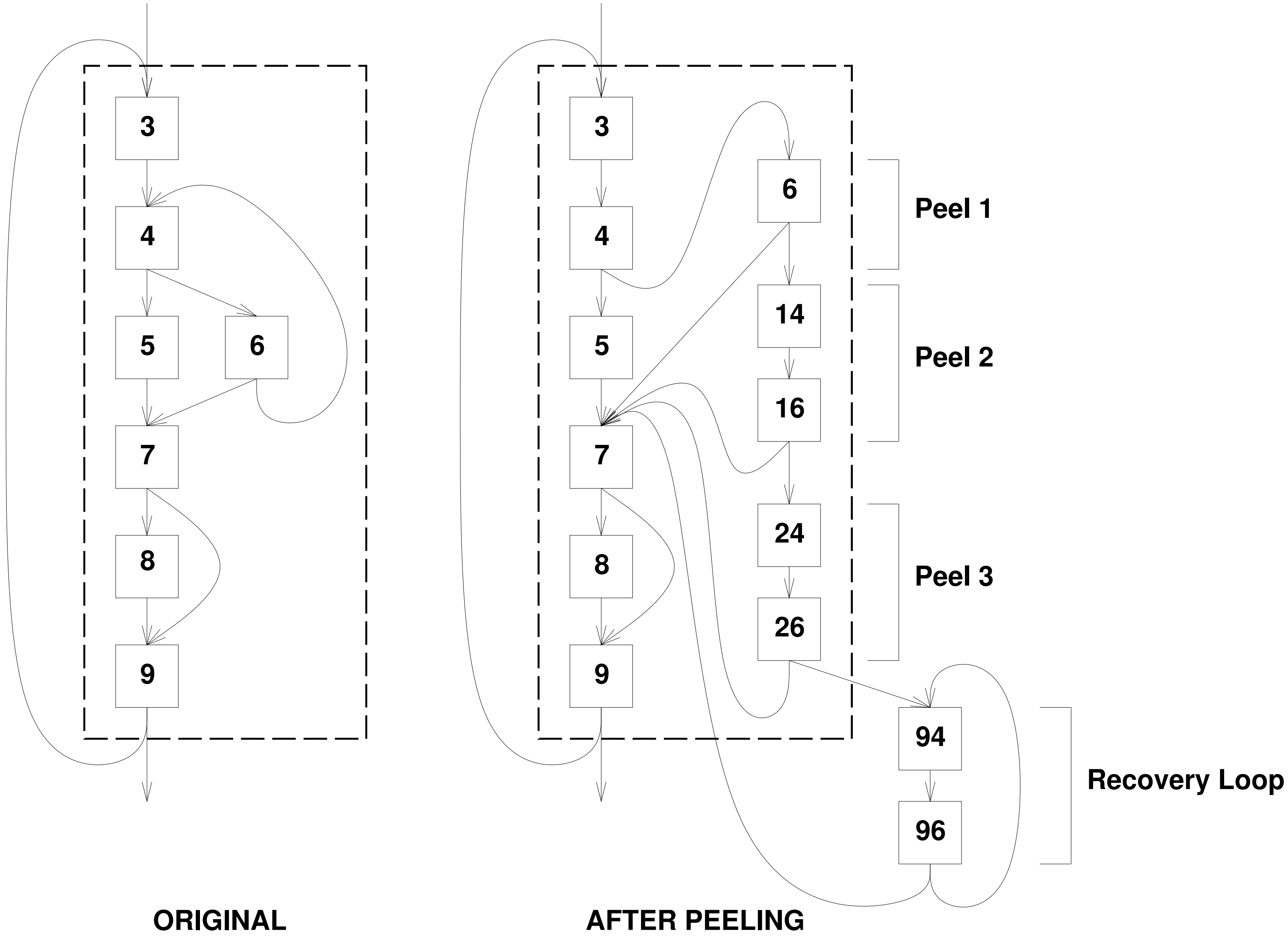
Inner Loop With Loop Profile Information

```
(cb 4 2500.000000 [(flow 1 5 400.000000)(flow 0 6 2100.000000)]
      <(iteration_header (i 3)(i 1))
       (iter_1 (f2 200)(f2 200))
       (iter_2 (f2 100)(f2 100))
       (iter_3 (f2 700)(f2 700))>)
  (op 15 ld_post_i <L> [(r 11 i)(r 35 i)] [(r 35 i)(i 0)(i 4)])
  (op 18 beq_fs [] [(r 11 i)(r 13 i)(cb 5)] <(loop_nest (i 2)>)
(cb 6 2100.000000 [(flow 1 4 1500.000000)(flow 0 7 600.000000)] )
  (op 25 add [(r 2 i)] [(r 2 i)(i 1)])
  (op 26 blt_fs [] [(r 2 i)(i 3)(cb 4)] <(loop_nest (i 2))(LB_inner)>)
  (op 80 jump [] [(cb 7)] <(loop_nest (i 2))(LE_inner)>)

(cb 5 ....

(cb 7 ....
```

Running Code Example Control Flow Graph



Inner Loop Peeled Twice and If-converted

```
pred_clear [(r 37 p)] []
mov [(r 2 i)] [(i 0)]
ld_i <L> [(r 13 i)] [(r 34 i)(i 0)]
add [(r 34 i)] [(r 34 i)(i 4)]
and [(r 7 i)] [(r 13 i)(i 7)]
mul [(r 8 i)] [(r 7 i)(i 12)]
add_u [(r 35 i)] [(r 8 i)(l _hash_table_element)]
ld_i [(r 11 i)] [(r 35 i)(i 0)] <(peel (i 1)(i 1))>
add [(r 35 i)] [(r 35 i)(i 4)] <(peel (i 1)(i 1))>
pred_eq [(r 37 p_ot)(r 40 p_uf)] [(r 11 i)(r 13 i)] <(peel (i 1)(i 1))>
add <(r 40 p)> [(r 2 i)] [(r 2 i)(i 1)] <(peel (i 1)(i 1))>
pred_lt <(r 40 p)> [(r 36 p_ut)] [(r 2 i)(i 3)] <(peel (i 1)(i 1))>
ld_i <(r 36 p)> [(r 11 i)] [(r 35 i)(i 0)] <(peel (i 2)(i 1))>
add <(r 36 p)> [(r 35 i)] [(r 35 i)(i 4)] <(peel (i 2)(i 1))>
pred_eq <(r 36 p)> [(r 37 p_ot)(r 38 p_uf)] [(r 11 i)(r 13 i)] <(peel (i 2)(i 1))>
add <(r 38 p)> [(r 2 i)] [(r 2 i)(i 1)] <(peel (i 2)(i 1))>
blt_fs <(r 38 p)> [] [(r 2 i)(i 3)(cb 4)] <(peel (i 2)(i 1))>
add_u <(r 37 p)> [(r 30 i)] [(r 30 i)(i 1)]
pred_eq [(r 39 p_ut)] [(r 2 i)(i 3)]
add_u <(r 39 p)> [(r 31 i)] [(r 31 i)(i 1)]
add [(r 1 i)] [(r 1 i)(i 1)]
blt_u_fs [] [(r 1 i)(r 21 i)(cb 18)]
jump [] [(cb 15)]
```

Important Peeled Loop Optimizations

- Classic loop optimizations should be applied to peeled loop iterations.

- Many such optimizations are performed by their equivalent straight line code counterparts. For example, invariant code removal is similar to common subexpression elimination.

- Accumulator expansion and induction variable elimination needed to be specifically implemented.
  - Originally a form of CPR applied to unrolled loops.
  - For each iteration we know how many iterations preceded it. This fact is used to calculate induction variable value for each peel iteration immediately.
  - Most important application is to eliminate dependencies between *address increment, load* chains.
  - One very important function in 008.espresso experiences a 7.0 time speedup when this optimization is used over when it is not used.

Induction Variable Elimination in the Code Example

```
1   r11 = mem(r35)
1   r35 = r35 + 4
2   p36 = (r11 == r13)
3   r11 = mem(r35)          (p36)
3   r35 = r35 + 4           (p36)
4   p37 = (r11 == r13)      (p36)
5   r11 = mem(r35)          (p37)
5   r35 = r35 + 4           (p37)
6   branch (r11 == r13)     (p37)
```

```
1   r135 = r35 + 4
1   r235 = r35 + 8
1   r335 = r35 + 12
1   r11 = mem(r35)
2   p36 = (r11 == r13)
3   r11 = mem(r135)         (p36)
4   p37 = (r11 == r13)      (p36)
5   r11 = mem(r235)         (p37)
6   branch (r11 == r13)     (p37)   ->   r35 = r335
```

```
1   r11 = mem(r35)
1   r35 = r35 + 4
2   p36 = (r11 == r13)
2   r111 = mem(r35)
3   r35 = r35 + 4           (p36)
3   p37 = (r111 == r13)
4   r211 = mem(r35)
4   r35 = r35 + 4           (p37)
5   branch (r211 == r13)    (p37)
```

```
1   r135 = r35 + 4
1   r235 = r35 + 8
1   r335 = r35 + 12
1   r11 = mem(r35)
2   p36 = (r11 == r13)
2   r111 = mem(r135)
3   p37 = (r111 == r13)     (p36)
2   r211 = mem(r235)
4   branch (r211 == r13)    (p37)   ->   r35 = r335
```

Performance of Code Example

Optimization	Cycles	IPC	Speedup
HB, Peeling, and AFS	4600	7.37	2.61
HB, Peeling	4800	7.40	2.50
HB, No Peeling	10800	2.05	1.11
Superblock	12000	2.09	1.00

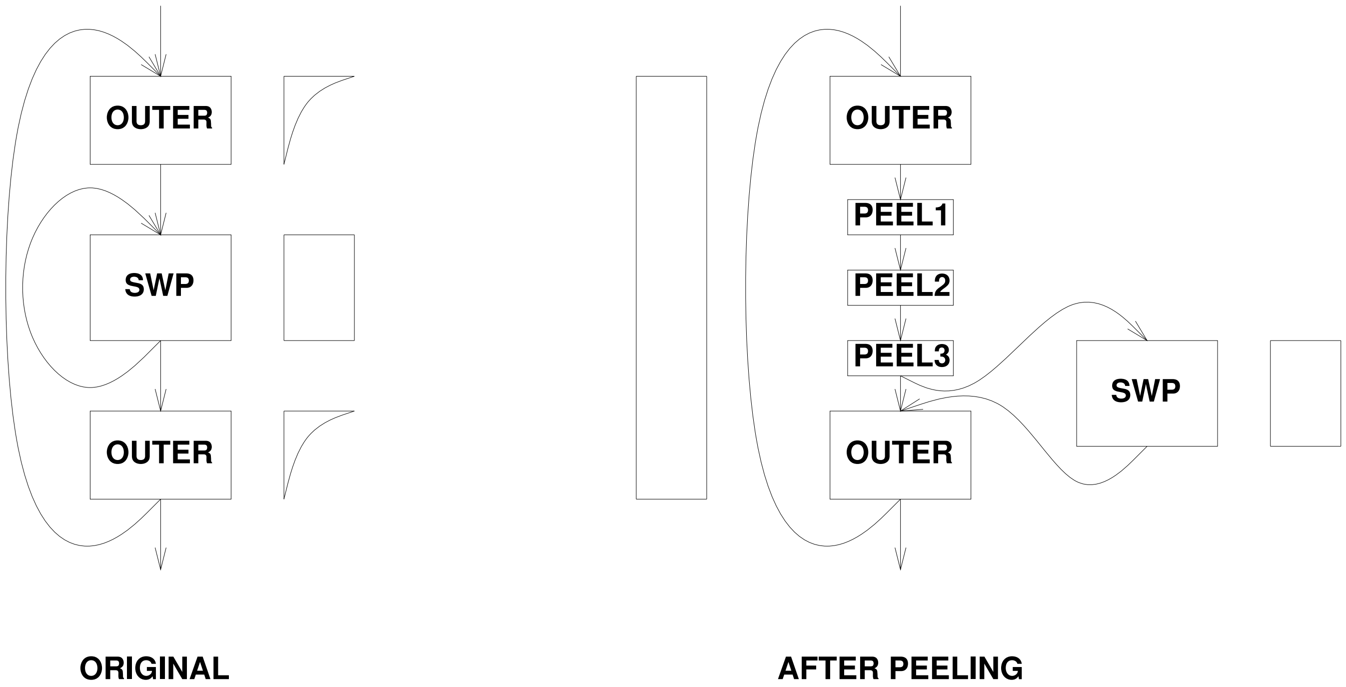
```
--------------+--------------------------------------------------
cb 3 100.0    |      1
 -> 11 0.0    |      2
 -> 4  100.0  |      3
              |  0   12     5L     4      7      8    10L   11L   14
              |  1   13L   20      -      -      -     -     -     -
              |  2    9     6B     -      -      -     -     -     -
              |  3   15     -      -      -      -     -     -     -
              |  4   17    16      -      -      -     -     -     -
              |  5   18    19      -      -      -     -     -     -
              |  6   21     -      -      -      -     -     -     -
--------------+--------------------------------------------------
cb 4 1000.0   |  0   22L   25L    28L    31L    34     -     -     -
 -> 24 200.0  |  1    -     -      -      -      -     -     -     -
 -> 25 0.0    |  2   23B    -      -      -      -     -     -     -
 -> 26 100.0  |  3   24B    -      -      -      -     -     -     -
 -> 27 0.0    |  4   26B    -      -      -      -     -     -     -
 -> 28 100.0  |  5   27B    -      -      -      -     -     -     -
 -> 29 600.0  |  6   29B    -      -      -      -     -     -     -
 -> 30 0.0    |  7   30B    -      -      -      -     -     -     -
 -> 4  0.0    |  8   32B   33      -      -      -     -     -     -
 -> 17 0.0    |  9   36    35B     -      -      -     -     -     -
--------------+--------------------------------------------------
cb 15 100.0   |  0   51S   52S     -      -      -     -     -     -
 -> 11 100.0  |
--------------+--------------------------------------------------
cb 11 100.0   |  0   53    54      -      -      -     -     -     -
              |
--------------+--------------------------------------------------
cb 21 400.0   |  0   57L   55     64     58      -     -     -     -
 -> 15 0.0    |  1   56B    -      -      -      -     -     -     -
 -> 4  400.0  |  2   59     -      -      -      -     -     -     -
              |  3   61    60      -      -      -     -     -     -
              |  4   62    63      -      -      -     -     -     -
              |  5   65    66J     -      -      -     -     -     -
--------------+--------------------------------------------------
cb 29 600.0   |  0   99L   95B    97     96    100   106     -     -
 -> 21 0.0    |  1   98B    -      -      -      -     -     -     -
 -> 15 100.0  |  2  101     -      -      -      -     -     -     -
 -> 4  500.0  |  3  103   102      -      -      -     -     -     -
              |  4  104   105      -      -      -     -     -     -
              |  5  107   108J     -      -      -     -     -     -
--------------+--------------------------------------------------
cb 24 200.0   |  0  109   110B   114L   112    111   121   115     -
 -> 21 200.0  |  1  113B    -      -      -      -     -     -     -
 -> 15 0.0    |  2  116     -      -      -      -     -     -     -
 -> 4  0.0    |  3  118   117      -      -      -     -     -     -
              |  4  119   120      -      -      -     -     -     -
              |  5  122   123J     -      -      -     -     -     -
--------------+--------------------------------------------------
cb 26 100.0   |  0  124   125B   129L   127    126   136   130     -
 -> 21 100.0  |  1  128B    -      -      -      -     -     -     -
 -> 15 0.0    |  2  131     -      -      -      -     -     -     -
 -> 4  0.0    |  3  133   132      -      -      -     -     -     -
              |  4  134   135      -      -      -     -     -     -
              |  5  137   138J     -      -      -     -     -     -
--------------+--------------------------------------------------
cb 28 100.0   |  0  139   140B   144L   142    141   151   145     -
 -> 21 100.0  |  1  143B    -      -      -      -     -     -     -
 -> 15 0.0    |  2  146     -      -      -      -     -     -     -
 -> 4  0.0    |  3  148   147      -      -      -     -     -     -
              |  4  149   150      -      -      -     -     -     -
              |  5  152   153J     -      -      -     -     -     -
--------------+--------------------------------------------------
```

```
--------------+--------------------------------------------------
cb 29 100.0   |      1
 -> 11 0.0    |      2
 -> 18 100.0  |      3
              |  0    4L     6      7      9L   10L   11    12    13
              |  1   14     15     16     17    18    19    20    21
              |  2   22     23     24     25    26    33    32    31
              |  3   30     29     28     27     8     5B    -     -
--------------+--------------------------------------------------
cb 18 200.0   |     34
 -> 15 0.0    |     35
 -> 15 100.0  |     36
 -> 15 0.0    |     37
 -> 15 0.0    |     38
 -> 15 0.0    |     39
 -> 15 0.0    |     40
 -> 15 0.0    |     41
 -> 18 100.0  |  0   43L    65L    86L   107L  128L  149L  170L  191L
 -> 15 0.0    |  1   42    211     64     85   106   127   148   169
              |  2   44     66     87    108   129   150   171   192
              |  3   46     45     68     67    89   110   131   152
              |  4   47     69    173     88   109   130   151   172
              |  5   49L    48     71L    70    90   111   132   153
              |  6   53L    57L    75L   174   194    79L  193    92L
              |  7   50     72    195    113L  134L  155L  176L   91
              |  8   52     74     51    112   133   154   175   197L
              |  9   54     76     93    114   135   156   177   196
              | 10   56     55     78     96L  117L  138L  159L  180L
              | 11   58    198     95    116   137   158   179    80
              | 12   59    201L    97    118   139   160   181   200
              | 13   61    100L   121L   142L  163L  184L   99   120
              | 14  202    141    162    183   205L   60    62    63B
              | 15   82     84B   101    122   204   143   164   185
              | 16  206    103    105B   124   190    73    94   115
              | 17  126B   145    136    157   178   199    77    98
              | 18  147B   166    119    140   161   182   203    81
              | 19  168B   187    102    123   144   165   186   207
              | 20  189B   210    208     83   104   125   146   167
              | 21  212B   188    209     -     -     -     -     -
--------------+--------------------------------------------------
cb 15 100.0   |  0  213    215    220    222   214   216   221   223
 -> 11 100.0  |  1  217    218    224    225    -     -     -     -
              |  2  219    226     -      -     -     -     -     -
              |  3  228S   227S    -      -     -     -     -     -
--------------+--------------------------------------------------
cb 11 100.0   |  0  229    230     -      -     -     -     -     -
              |
--------------+--------------------------------------------------
```

Hyperblock Re-entry

Some hyperblocks are more effective when recovery loop can jump back into hyperblock.

- Tail duplication causes extra code expansion.
- Tail duplicated code is generally less optimal than in trace code. Usually less overlapping possible with other code.

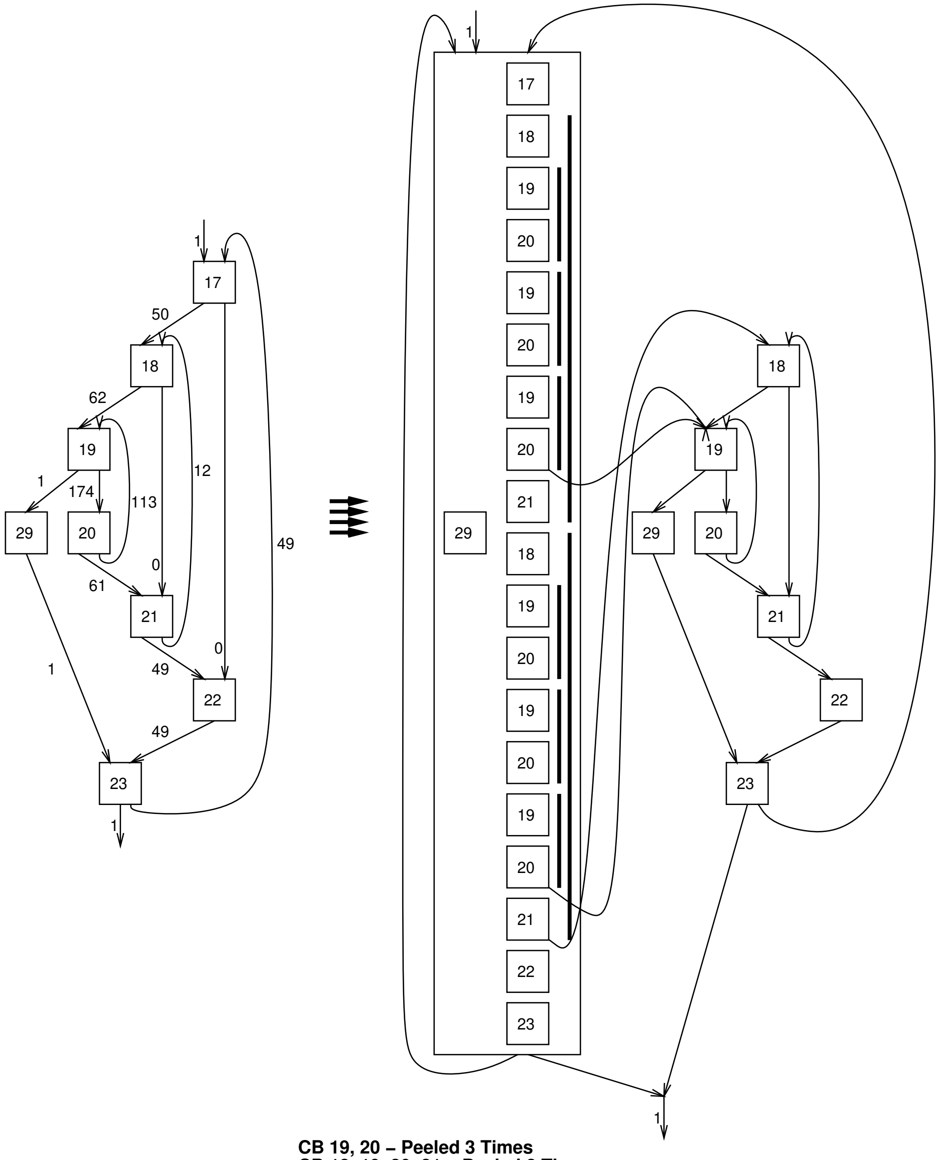


Conclusions And Future Work

- Conclusions:
  - Loop peeling increases the effectiveness of hyperblock techniques.
  - Classical loop optimizations should be applied to peeled loops.
- Future Work:
  - Consider loops for peeling when outer loop is sparse.
  - Improve current heuristic to consider dependence height information.
  - Evaluate effectiveness of returning to the hyperblock after loop recovery.
  - Explore other hyperblock optimizations such as node splitting and partial reverse if-conversion.

Nested Loop Peeling Example

SIEV – f_2



CB 19, 20 – Peeled 3 Times
CB 18, 19, 20, 21 – Peeled 2 Times