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

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

```c
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
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
- ...

Loop Representation in LLVM

Preheader
A single edge to the header of the loop from outside of the loop.

Header
Single entry point to the loop that dominates all other blocks in the loop.

Exiting Block
The block within the loop that has successors outside of the loop. If multiple blocks have successors, this is null.

Latch
Block that contains the branch back to the loop header.

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

LLVM Developers’ Meeting 2018
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:

1. **Alias Analysis**
   - Test if two memory locations alias each other

2. **Dependence Info**
   - Uses the depends interface from Dependence Info

3. **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

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Candidates for Fusion</th>
<th>Loops Fused</th>
</tr>
</thead>
<tbody>
<tr>
<td>perlbench</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>bzip2</td>
<td>21</td>
<td>1</td>
</tr>
<tr>
<td>gcc</td>
<td>50</td>
<td>8</td>
</tr>
<tr>
<td>namd</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>gobmk</td>
<td>96</td>
<td>7</td>
</tr>
<tr>
<td>dealII</td>
<td>355</td>
<td>0</td>
</tr>
<tr>
<td>soplex</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>povray</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>hammmer</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>h264ref</td>
<td>159</td>
<td>2</td>
</tr>
<tr>
<td>lbm</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>astar</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>sphinx3</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

### SPEC 2017

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Candidates for Fusion</th>
<th>Loops Fused</th>
</tr>
</thead>
<tbody>
<tr>
<td>perlbench_r</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>gcc_r</td>
<td>114</td>
<td>2</td>
</tr>
<tr>
<td>namd_r</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>parest_r</td>
<td>137</td>
<td>1</td>
</tr>
<tr>
<td>povray_r</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>lbm_r</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>omnetpp_r</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>x264_r</td>
<td>81</td>
<td>6</td>
</tr>
<tr>
<td>blender_r</td>
<td>259</td>
<td>6</td>
</tr>
<tr>
<td>deepsjeng_r</td>
<td>34</td>
<td>0</td>
</tr>
<tr>
<td>imagick_r</td>
<td>45</td>
<td>5</td>
</tr>
<tr>
<td>nab_r</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>xz_r</td>
<td>18</td>
<td>1</td>
</tr>
</tbody>
</table>
## Reasons for not fusing

### SPEC 2006

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

### SPEC 2017

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

### SPEC 2006

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

### SPEC 2017

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