LARA Tutorial

3. Code Transformations

Tiago Carvalho, Pedro Pinto, João Bispo, Ricardo Nobre, Luís Reis, and João M.P. Cardoso

University of Porto, FEUP, Porto, Portugal

April 20th, 2016

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

ANTAREX 10^{18}

SPeCS
Special-Purpose Computing Systems, languages and tools

Objectives

- Present code transformation concepts in LARA

- Show actions for
  - Optimization
  - Specialization
  - Type definition

Some Information

MANET:

- LARA + Cetus*
- ANSI C
- http://specs.fe.up.pt/tools/manet/

- The presented examples
  - Dijkstra from MiBench
  - Disparity from San Diego Vision Benchmark Suite

*Cetus compiler: http://cetus.ecn.purdue.edu/

3.1 Simple Loop Transformations – Goal

- Show how to use actions in LARA
  - Present the actions available in MANET
  - The *exec* keyword
- Use loop unroll as an example and parameterize it

3.1 Simple Loop Unroll – Strategy

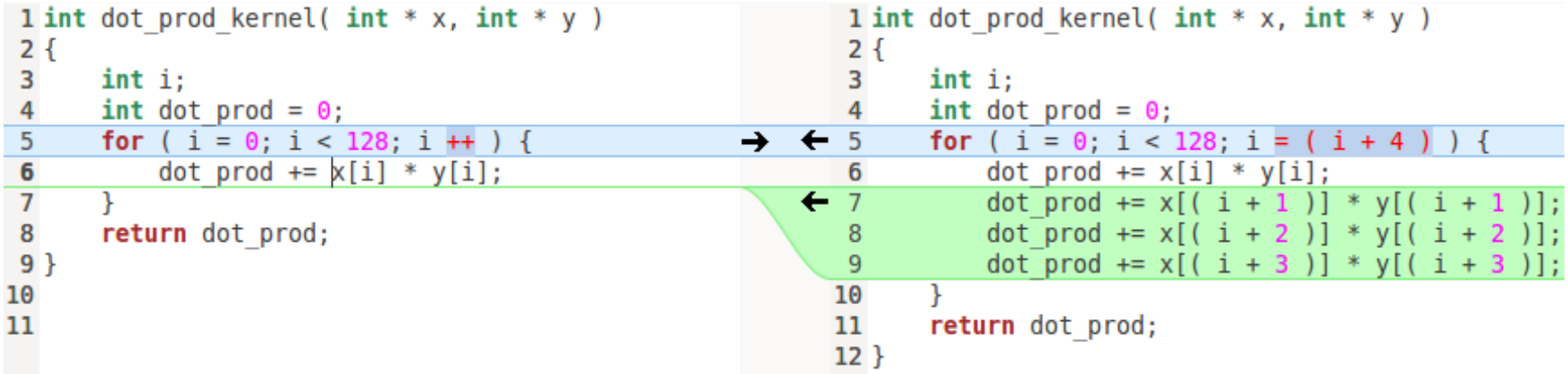
- Capture relevant loops
- Filter them using attributes
  - *is_innermost*
  - *num_iterations*
- *exec* Unroll with different parameters

```
aspectdef SimpleLoopUnroll

  input funcName, factor end

  select function{funcName}.loop end
  apply
    exec Unroll(factor); // 0 for full
  end
  condition
    $loop.is_innermost
  end
end
```

3.1 Simple Loop Transformations – Diff



```
1  int dot_prod_kernel( int * x, int * y )
2  {
3      int i;
4      int dot_prod = 0;
5      for ( i = 0; i < 128; i ++ ) {
6          dot_prod += x[i] * y[i];
7      }
8      return dot_prod;
9  }
10
11
```

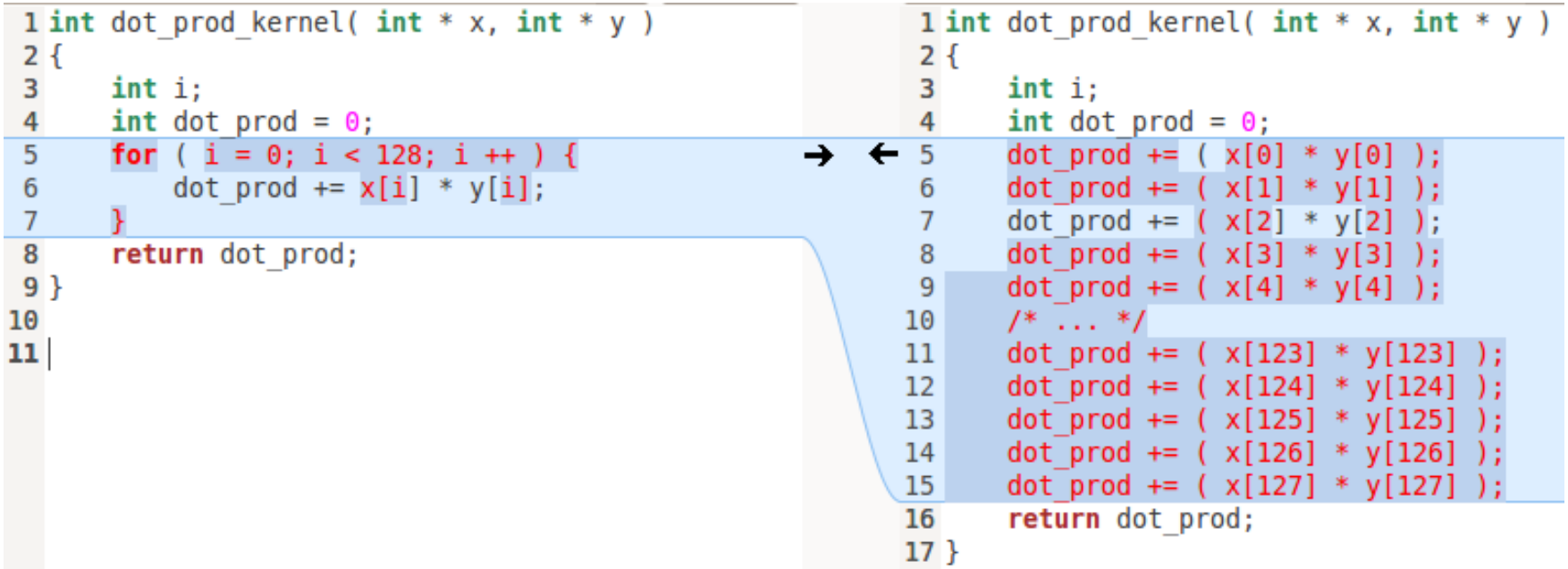
```
1  int dot_prod_kernel( int * x, int * y )
2  {
3      int i;
4      int dot_prod = 0;
5      for ( i = 0; i < 128; i = ( i + 4 ) ) {
6          dot_prod += x[i] * y[i];
7          dot_prod += x[( i + 1 )] * y[( i + 1 )];
8          dot_prod += x[( i + 2 )] * y[( i + 2 )];
9          dot_prod += x[( i + 3 )] * y[( i + 3 )];
10     }
11     return dot_prod;
12 }
```

Unroll with factor 4 and known iteration count

Main LARA code:

```
exec Unroll(factor);
```

3.1 Simple Loop Transformations – Diff



```
 1 int dot_prod_kernel( int * x, int * y )
 2 {
 3     int i;
 4     int dot_prod = 0;
 5     for ( i = 0; i < 128; i ++ ) {
 6         dot_prod += x[i] * y[i];
 7     }
 8     return dot_prod;
 9 }
10
11
```

```
 1 int dot_prod_kernel( int * x, int * y )
 2 {
 3     int i;
 4     int dot_prod = 0;
 5     dot_prod += ( x[0] * y[0] );
 6     dot_prod += ( x[1] * y[1] );
 7     dot_prod += ( x[2] * y[2] );
 8     dot_prod += ( x[3] * y[3] );
 9     dot_prod += ( x[4] * y[4] );
10     /* ... */
11     dot_prod += ( x[123] * y[123] );
12     dot_prod += ( x[124] * y[124] );
13     dot_prod += ( x[125] * y[125] );
14     dot_prod += ( x[126] * y[126] );
15     dot_prod += ( x[127] * y[127] );
16     return dot_prod;
17 }
```

Unroll with factor 0 (full unroll) and known iteration count

3.1 Simple Loop Transformations – Diff

```
1  int dot_prod_kernel( int * x, int * y )
2  {
3      int i;
4      int dot_prod = 0;
5      for ( i = 0; i < SIZE; i ++ ) {
6          dot_prod += x[i] * y[i];
7      }
8      return dot_prod;
9  }
10
11
```

```
1  int dot_prod_kernel( int * x, int * y )
2  {
3      int i;
4      int dot_prod = 0;
5      for ( i = 0; i < ( -1 + SIZE ) + ( -1 * ( ( -1 + SIZE ) % 4 ) ); i = i + 4 ) {
6          dot_prod += x[i] * y[i];
7          dot_prod += x[( i + 1 )] * y[( i + 1 )];
8          dot_prod += x[( i + 2 )] * y[( i + 2 )];
9          dot_prod += x[( i + 3 )] * y[( i + 3 )];
10     }
11     /*
12     Epilogue
13     */
14     for ( i = ( -1 + SIZE ) + ( -1 * ( ( -1 + SIZE ) % 4 ) ); i < SIZE; i ++ ) {
15         dot_prod += ( x[i] * y[i] );
16     }
17     return dot_prod;
18 }
```

Unroll with factor 4 and unknown iteration count

3.1 Simple Loop Unroll – Another Strategy

- Account for the number of iterations
- Insert a simple conditional statement
  - Full unroll if it performs up to 8 iterations
- More sophisticated aspects
  - Attributes
  - Metrics
  - Parameters

```
aspectdef SimpleLoopUnroll

  input funcName, factor end

  select function{funcName}.loop end
  apply
    if($loop.num_iterations <= 8)
      exec Unroll(0);
    else
      exec Unroll(factor);
  end
  condition
    $loop.is_innermost
  end
end
```

3.2 Change Data Types – Goal

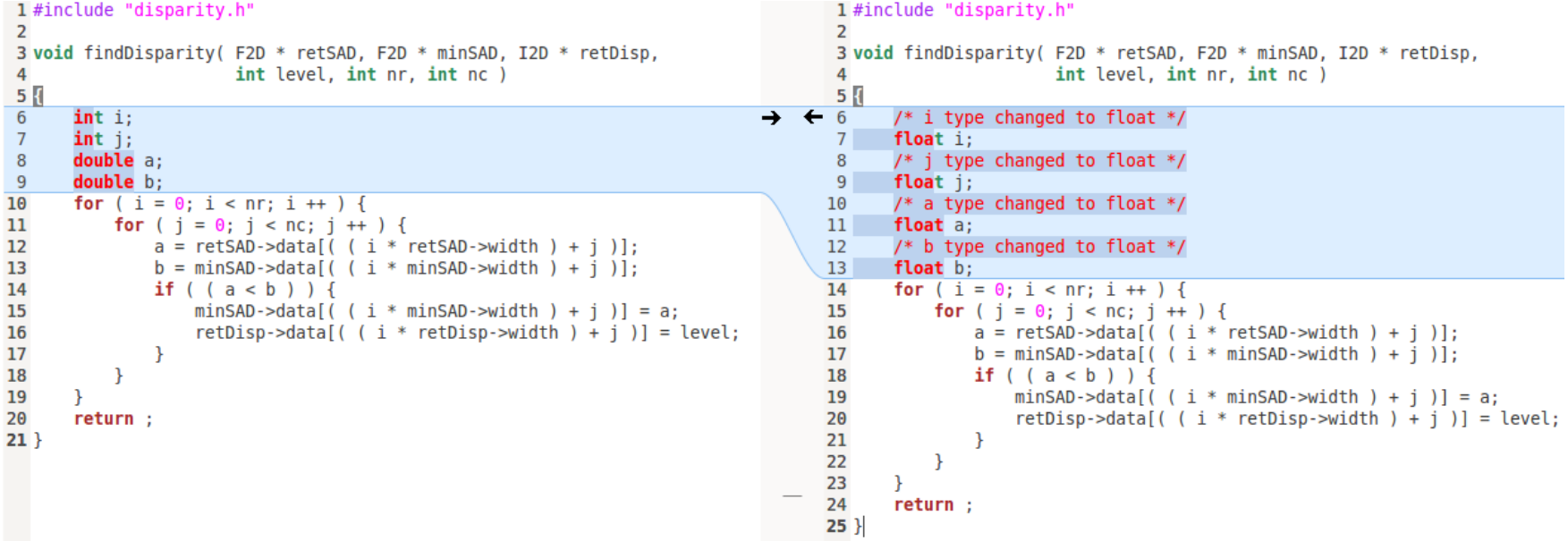
- Change the types of certain variables
  - Using the *def* action
- Filter unwanted variables
  - type, operation, location, name, …
- Can also change types of function returns and parameters
- Could be based on a variable range analysis

3.2 Change Data Types – Strategy

- Prepare variable declarations
  - *exec* SingleDeclarator()
- Select target variables
  - Searching for declarations inside functions
- Use the *def* action to define the value of the *native_type* attribute

```
select function{funcName}.decl end
apply
  def native_type = newT;
end
```

3.2 Change Data Types – Diff



```
#include "disparity.h"

void findDisparity( F2D * retSAD, F2D * minSAD, I2D * retDisp,
                    int level, int nr, int nc )
{
    int i;
    int j;
    double a;
    double b;
    for ( i = 0; i < nr; i ++ ) {
        for ( j = 0; j < nc; j ++ ) {
            a = retSAD->data[( ( i * retSAD->width ) + j )];
            b = minSAD->data[( ( i * minSAD->width ) + j )];
            if ( ( a < b ) ) {
                minSAD->data[( ( i * minSAD->width ) + j )] = a;
                retDisp->data[( ( i * retDisp->width ) + j )] = level;
            }
        }
    }
    return ;
}
```

```
#include "disparity.h"

void findDisparity( F2D * retSAD, F2D * minSAD, I2D * retDisp,
                    int level, int nr, int nc )
{
    /* i type changed to float */
    float i;
    /* j type changed to float */
    float j;
    /* a type changed to float */
    float a;
    /* b type changed to float */
    float b;
    for ( i = 0; i < nr; i ++ ) {
        for ( j = 0; j < nc; j ++ ) {
            a = retSAD->data[( ( i * retSAD->width ) + j )];
            b = minSAD->data[( ( i * minSAD->width ) + j )];
            if ( ( a < b ) ) {
                minSAD->data[( ( i * minSAD->width ) + j )] = a;
                retDisp->data[( ( i * retDisp->width ) + j )] = level;
            }
        }
    }
    return ;
}
```

Main LARA code:

```
def native_type = newT;
```

Without control variable filtering

3.2 Change Data Types – Avoid Control Variables

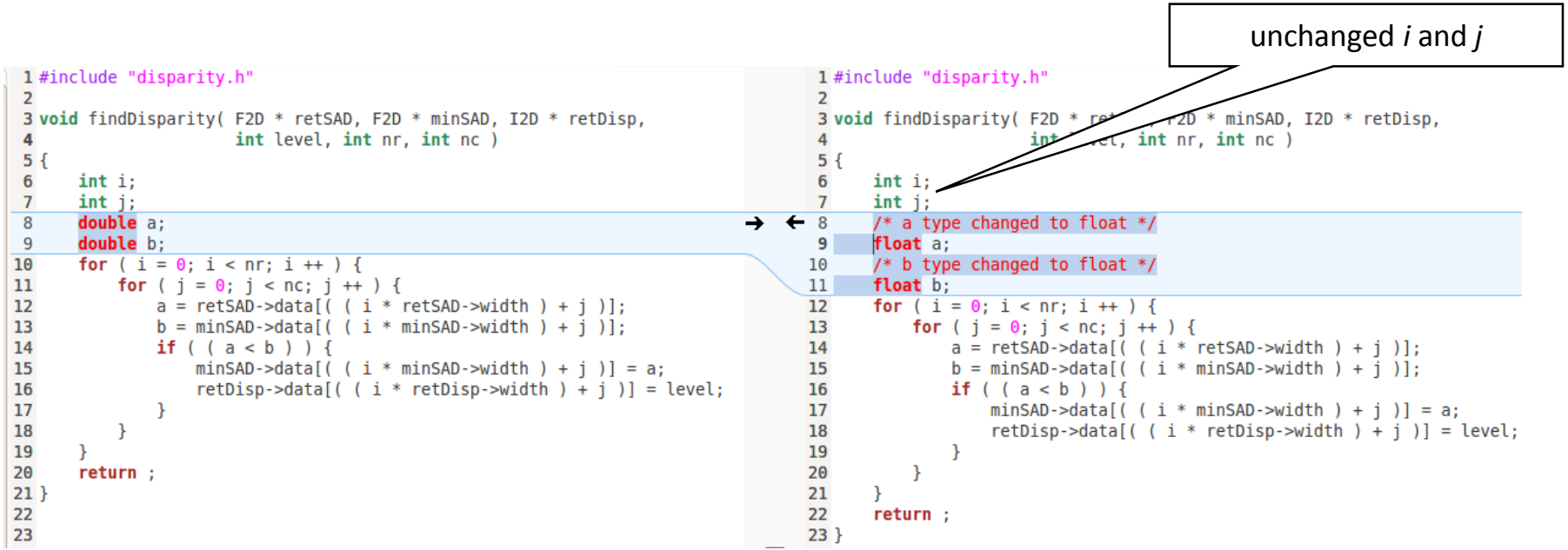
- Use join point attributes to filter the results
  - Select all loops in target function
  - Collect their control variables in a set
- How to avoid loop control variables
  - This way
  - Common names (i, j, k, …)
  - *in_loop_header* attribute
  - …

```
var loopControlVars = new Set();

select function{funcName}.loop end
apply
  loopControlVars.add($loop.control_var);
end

select function{funcName}.decl end
apply
  def native_type = newT;
end
condition
  !loopControlVars.contains($decl.name)
end
```

3.2 Change Data Types – Diff



With control variable filtering

3.3 Cloning and Transformations – Goal

- Change a program for possible specialization / adaptation
- Generate several versions of a target function
  - Using the *Clone* action
- Apply transformations to the clones
- Add a way of selecting the newly generated versions

3.3 Cloning and Transformations – Strategy

- Clone a critical function and assign a new name to the clone
- The *Clone* action generates new copies after the selected function

```
var tiled = funcName + '_tiled';

select function{funcName} end
apply
  exec Clone(tiled);
end
```

3.3 Cloning and Transformations – Diff

```
1 #include "disparity.h"
2
3 void computeSAD( I2D * Ileft, I2D * Iright_moved, F2D * SAD )
4 {
5     int rows, cols, i, j, diff;
6     rows = Ileft->height;
7     cols = Ileft->width;
8     for ( i = 0; i < rows; i ++ ) {
9         for ( j = 0; j < cols; j ++ ) {
10            diff = Ileft->data[( ( i * Ileft->width ) + j )] -
11                   Iright_moved->data[( ( i * Iright_moved->width ) + j )];
12            SAD->data[( ( i * SAD->width ) + j )] = ( diff * diff );
13        }
14    }
15    return ;
16 }
```

```
1 #include "disparity.h"
2
3 void computeSAD_tiled( I2D * Ileft, I2D * Iright_moved, F2D * SAD )
4 {
5     int rows, cols, i, j, diff;
6     rows = Ileft->height;
7     cols = Ileft->width;
8     for ( i = 0; i < rows; i ++ ) {
9         for ( j = 0; j < cols; j ++ ) {
10            diff = Ileft->data[( ( i * Ileft->width ) + j )] -
11                   Iright_moved->data[( ( i * Iright_moved->width ) + j )];
12            SAD->data[( ( i * SAD->width ) + j )] = ( diff * diff );
13        }
14    }
15    return ;
16 }
17
18 void computeSAD( I2D * Ileft, I2D * Iright_moved, F2D * SAD )
19 {
20    int rows, cols, i, j, diff;
21    rows = Ileft->height;
22    cols = Ileft->width;
23    for ( i = 0; i < rows; i ++ ) {
24        for ( j = 0; j < cols; j ++ ) {
25            diff = Ileft->data[( ( i * Ileft->width ) + j )] -
26                   Iright_moved->data[( ( i * Iright_moved->width ) + j )];
27            SAD->data[( ( i * SAD->width ) + j )] = ( diff * diff );
28        }
29    }
30    return ;
31 }
```

Main LARA code:

```
exec Clone(tiled);
```

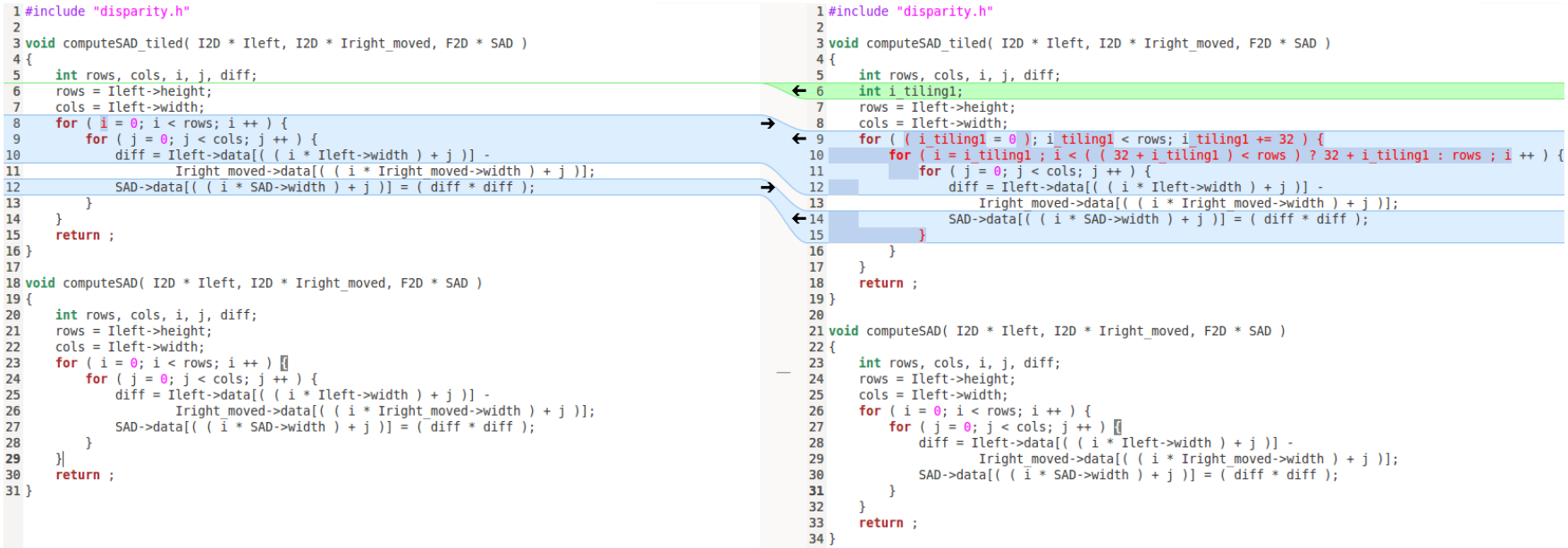
3.3 Cloning and Transformations – Strategy

- Select functions with the new names

- Transform them
  - Loop Tiling
  - Loop Interchange
  - …

```
select function{tiled}.loop end
apply
  exec Tile(32);
end
condition
  $loop.is_outermost
end
```

3.3 Cloning and Transformations – Diff



```
#include "disparity.h"

void computeSAD_tiled( I2D * Ileft, I2D * Iright_moved, F2D * SAD )
{
    int rows, cols, i, j, diff;
    rows = Ileft->height;
    cols = Ileft->width;
    for ( i = 0; i < rows; i ++ ) {
        for ( j = 0; j < cols; j ++ ) {
            diff = Ileft->data[( i * Ileft->width ) + j )] -
                    Iright_moved->data[( i * Iright_moved->width ) + j )];
            SAD->data[( i * SAD->width ) + j )] = ( diff * diff );
        }
    }
    return ;
}

void computeSAD( I2D * Ileft, I2D * Iright_moved, F2D * SAD )
{
    int rows, cols, i, j, diff;
    rows = Ileft->height;
    cols = Ileft->width;
    for ( i = 0; i < rows; i ++ ) {
        for ( j = 0; j < cols; j ++ ) {
            diff = Ileft->data[( i * Ileft->width ) + j )] -
                    Iright_moved->data[( i * Iright_moved->width ) + j )];
            SAD->data[( i * SAD->width ) + j )] = ( diff * diff );
        }
    }
    return ;
}
```

```
#include "disparity.h"

void computeSAD_tiled( I2D * Ileft, I2D * Iright_moved, F2D * SAD )
{
    int rows, cols, i, j, diff;
    int i_tiling1;
    rows = Ileft->height;
    cols = Ileft->width;
    for ( i_tiling1 = 0 ); i_tiling1 < rows; i_tiling1 += 32 ) {
        for ( i = i_tiling1 ; i < ( ( 32 + i_tiling1 ) < rows ) ? 32 + i_tiling1 : rows ; i ++ ) {
            for ( j = 0; j < cols; j ++ ) {
                diff = Ileft->data[( i * Ileft->width ) + j )] -
                        Iright_moved->data[( i * Iright_moved->width ) + j )];
                SAD->data[( i * SAD->width ) + j )] = ( diff * diff );
            }
        }
    }
    return ;
}

void computeSAD( I2D * Ileft, I2D * Iright_moved, F2D * SAD )
{
    int rows, cols, i, j, diff;
    rows = Ileft->height;
    cols = Ileft->width;
    for ( i = 0; i < rows; i ++ ) {
        for ( j = 0; j < cols; j ++ ) {
            diff = Ileft->data[( i * Ileft->width ) + j )] -
                    Iright_moved->data[( i * Iright_moved->width ) + j )];
            SAD->data[( i * SAD->width ) + j )] = ( diff * diff );
        }
    }
    return ;
}
```

Main LARA code:

```
exec Tile(32);
```

3.3 Cloning and Transformations – Strategy

- Find calls to the old function and add new logic
- Inserting simple switch with a user-generated function

```
select call{funcName} end
apply
  var originalCode = $call.code + ';';
  var tiledCode = originalCode.replace(funcName, tiled);
  insert before '/*';
  insert after '*/';

  insert after %{
    switch (get_best_version(/*...*/)) {
      case 0: [[originalCode]] break;
      case 1: [[tiledCode]] break;
      /* other user versions */
      default: [[originalCode]] break;
    }
  }%;
end
```

3.3 Cloning and Transformations – Diff

```
1 #include "disparity.h"
2
3 void correlateSAD_2D( I2D * Ileft, I2D * Iright, I2D * Iright_moved, int win_sz,
4                       int disparity, F2D * SAD, F2D * integralImg, F2D * retSAD )
5 {
6     int rows, cols;
7     int i, j, endRM;
8     I2D * range;
9     range = iMallocHandle( 1, 2 );
10    range->data[( ( 0 * range->width ) + 0 )] = 0;
11    range->data[( ( 0 * range->width ) + 1 )] = disparity;
12    rows = Iright_moved->height;
13    cols = Iright_moved->width;
14    for ( i = 0; i < ( rows * cols ); i ++ ) {
15        Iright_moved->data[i] = 0;
16    }
17    padarray4( Iright, range, ( - 1 ), Iright_moved );
18    computeSAD( Ileft, Iright_moved, SAD );
19    integralImage2D2D( SAD, integralImg );
20    finalSAD( integralImg, win_sz, retSAD );
21    iFreeHandle( range );
22    return ;
23 }
```

```
1 #include "disparity.h"
2
3 void correlateSAD_2D( I2D * Ileft, I2D * Iright, I2D * Iright_moved, int win_sz,
4                       int disparity, F2D * SAD, F2D * integralImg, F2D * retSAD )
5 {
6     int rows, cols;
7     int i, j, endRM;
8     I2D * range;
9     range = iMallocHandle( 1, 2 );
10    range->data[( ( 0 * range->width ) + 0 )] = 0;
11    range->data[( ( 0 * range->width ) + 1 )] = disparity;
12    rows = Iright_moved->height;
13    cols = Iright_moved->width;
14    for ( i = 0; i < ( rows * cols ); i ++ ) {
15        Iright_moved->data[i] = 0;
16    }
17    padarray4( Iright, range, ( - 1 ), Iright_moved );
18    /*
19    computeSAD(Ileft, Iright_moved, SAD);
20    */
21
22    switch ( get_best_version( /*...*/ ) ) {
23    case 0:
24        computeSAD( Ileft, Iright_moved, SAD );
25        break;
26    case 1:
27        computeSAD_tiled( Ileft, Iright_moved, SAD );
28        break;
29        /* other user versions */
30    default:
31        computeSAD( Ileft, Iright_moved, SAD );
32        break;
33    }
34
35    integralImage2D2D( SAD, integralImg );
36    finalSAD( integralImg, win_sz, retSAD );
37    iFreeHandle( range );
38    return ;
39 }
```

Main LARA code:

```
insert after %{
    switch (get_best_version(/*...*/))
    /*...*/
}%;
```

Takeaway Points

- Code transformations in LARA rely on
  - *def*
  - *insert*
  - All others actions, called with *exec*
- Enhance simple transformation strategies with attribute filtering
- Strategies can be used as standalone or prepare code for
  - Other aspects
  - Other tools
  - User development