LARA Tutorial

2. Dynamic Analysis

through Code Instrumentation

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

- Further usage of LARA
- Capture and report runtime information
- Use code instrumentation
  - *insert* action

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/

2.1 Timing Code Fragments – Goals

- Measure the execution time of code points
- Print current time and total elapsed time

Expected output:

```
...
current time of square: 704593.42us
current time of square: 716286.09us
current time of square: 727852.52us
current time of square: 739440.62us
elapsed time for square: 739440.62us
```

2.1 Timing Code Fragments – Strategy

- Instantiate a timing monitor
- Add C code to setup monitor

- Select target code to measure
  - Surround with start/pause statements
  - Add printing code for current time

- Add printing code before main return

```
aspectdef InstrumentApplication

  input functionName, timer end

  select call{functionName} end
  apply
    insert before '[[timer.start]]';
    insert after  '[[timer.pause]]';
  end
end
```

2.1 Timing Code Fragments – Diff

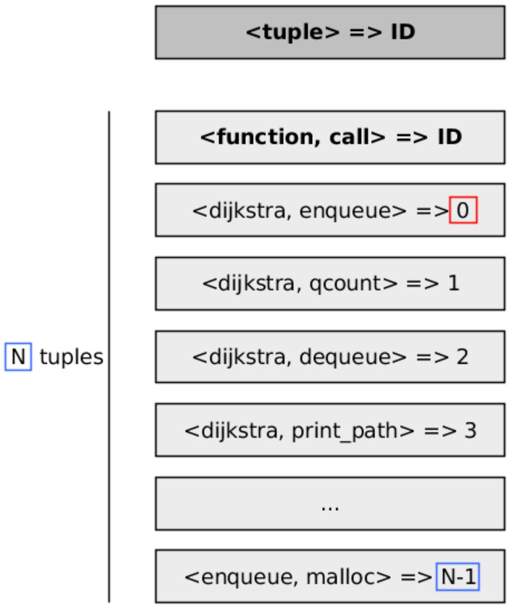
```
#include <stdio.h>
int square( int c )
{
    return ( c * c );
}

int sum_squares( int a, int b )
{
    int a2;
    int b2;
    a2 = square( a );
    b2 = square( b );
    return ( a2 + b2 );
}

int main()
{
    int a = 10;
    int b = 25;
    int c;
    c = sum_squares( a, b );
    printf( "Result: %d\n", c );
    return 0;
}
```

```
#include <stdio.h>
#include "timer.h"
Timer * tim = NULL;
int square( int c )
{
    return ( c * c );
}

int sum_squares( int a, int b )
{
    int a2;
    int b2;

    timer_start( tim );
    a2 = square( a );
    timer_pause( tim );
    printf( "current elapsed time of square: %.2fus\n", timer_get_time( tim ) );

    timer_start( tim );
    b2 = square( b );
    timer_pause( tim );
    printf( "current elapsed time of square: %.2fus\n", timer_get_time( tim ) );

    return ( a2 + b2 );
}

int main()
{
    int a = 10;
    int b = 25;
    int c;
    tim = timer_init();
    c = sum_squares( a, b );
    printf( "Result: %d\n", c );
    printf( "elapsed time for square: %.2fus\n", timer_get_time( tim ) );
    timer_destroy( tim );
    return 0;
}
```

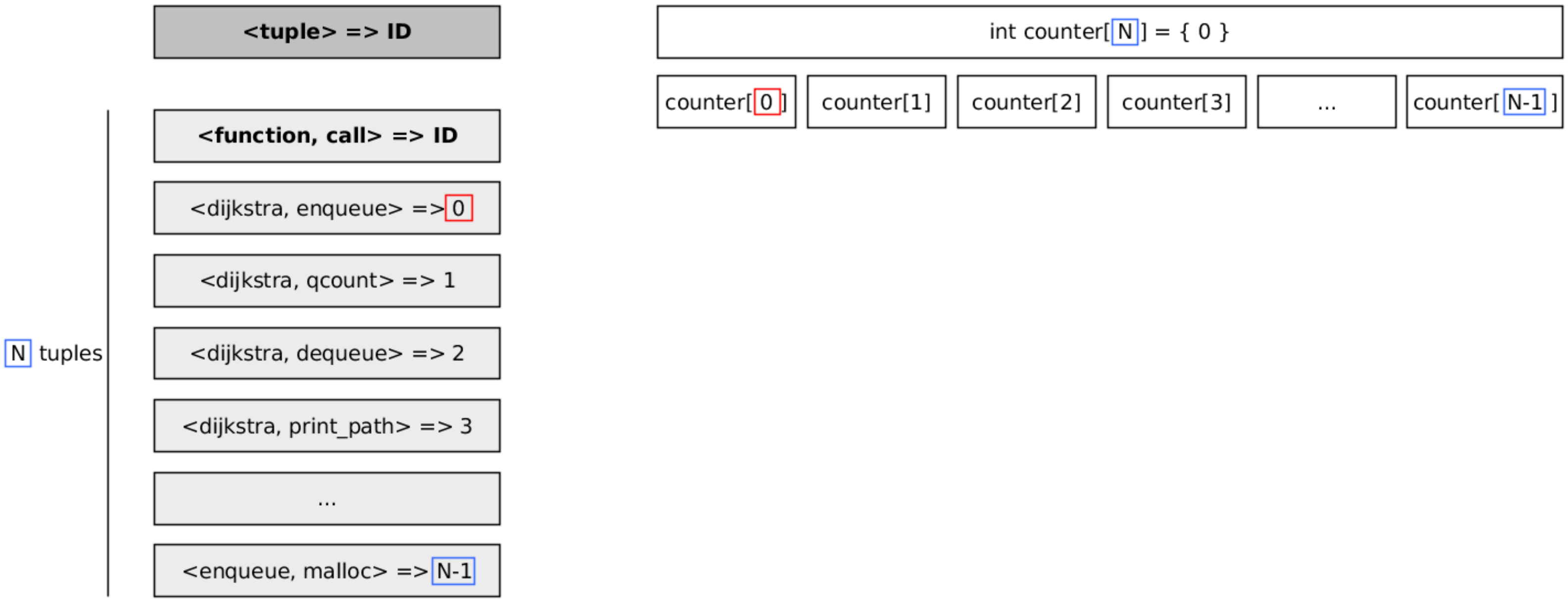
Main LARA code:

```
insert before '[[timer.start]]';
insert after  '[[timer.pause]]';
```

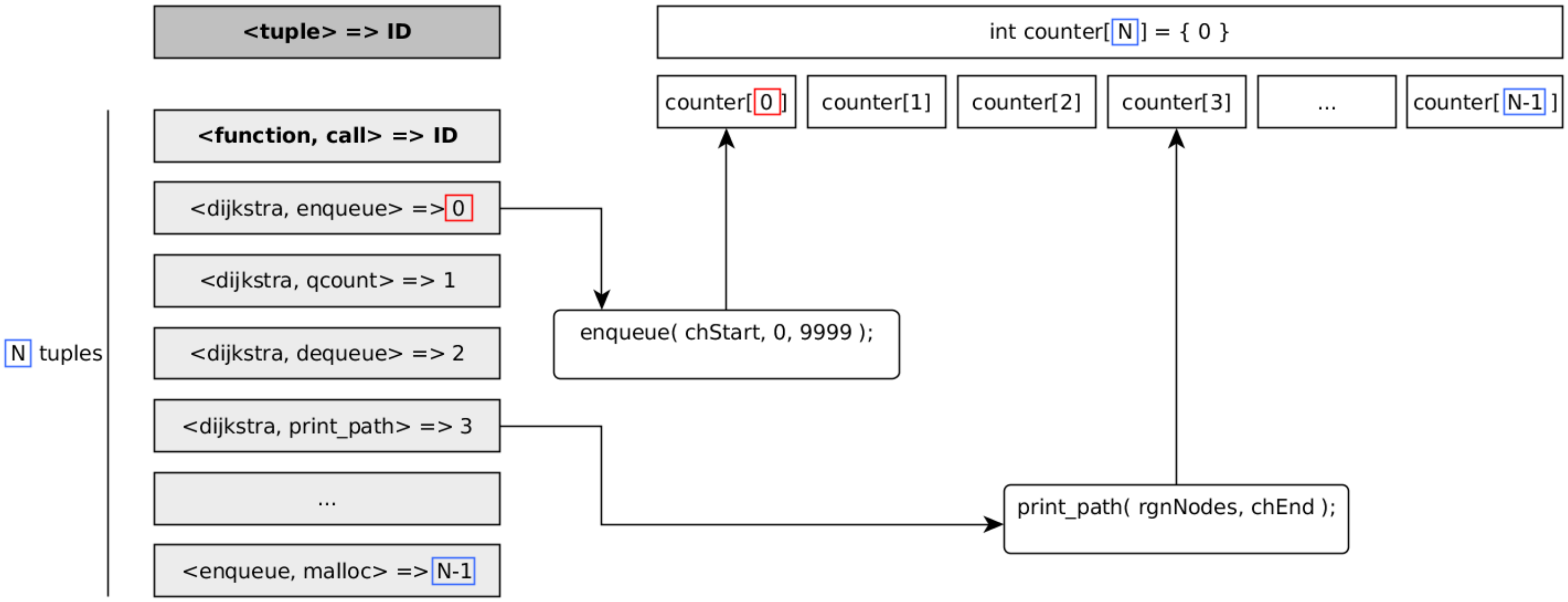
Capturing and Using Tuples



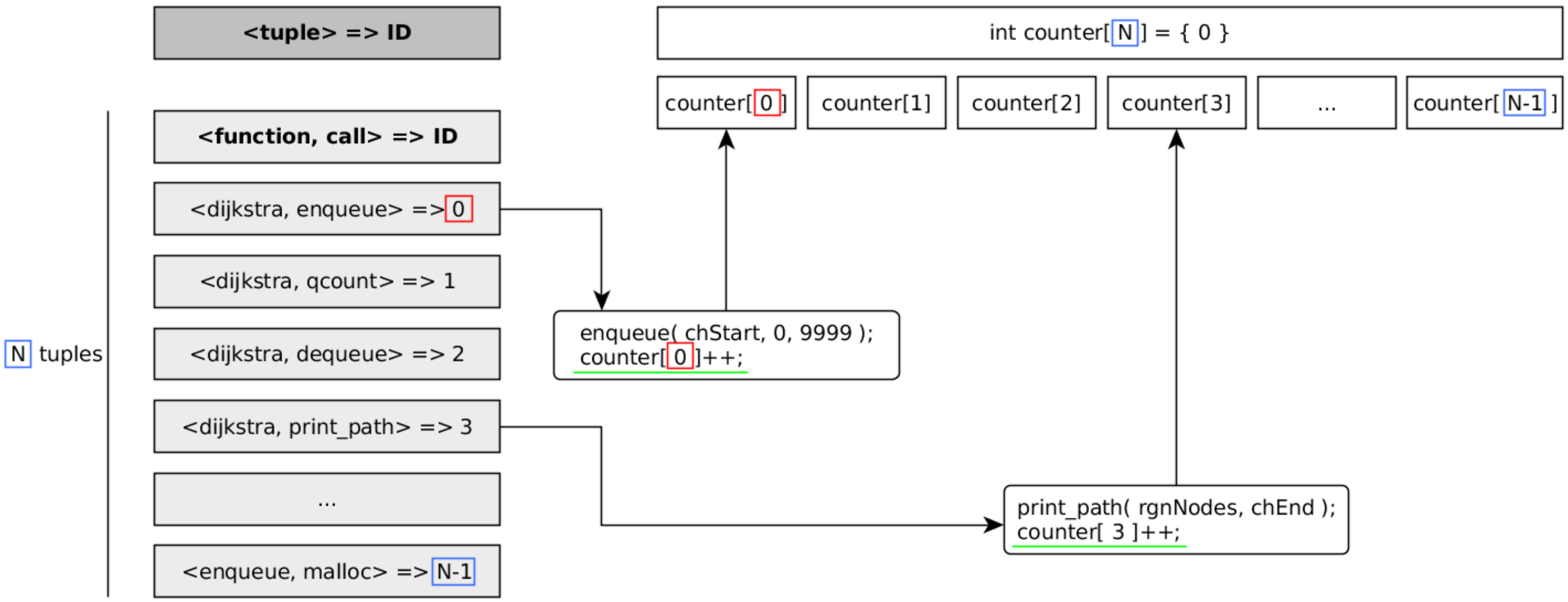
Capturing and Using Tuples



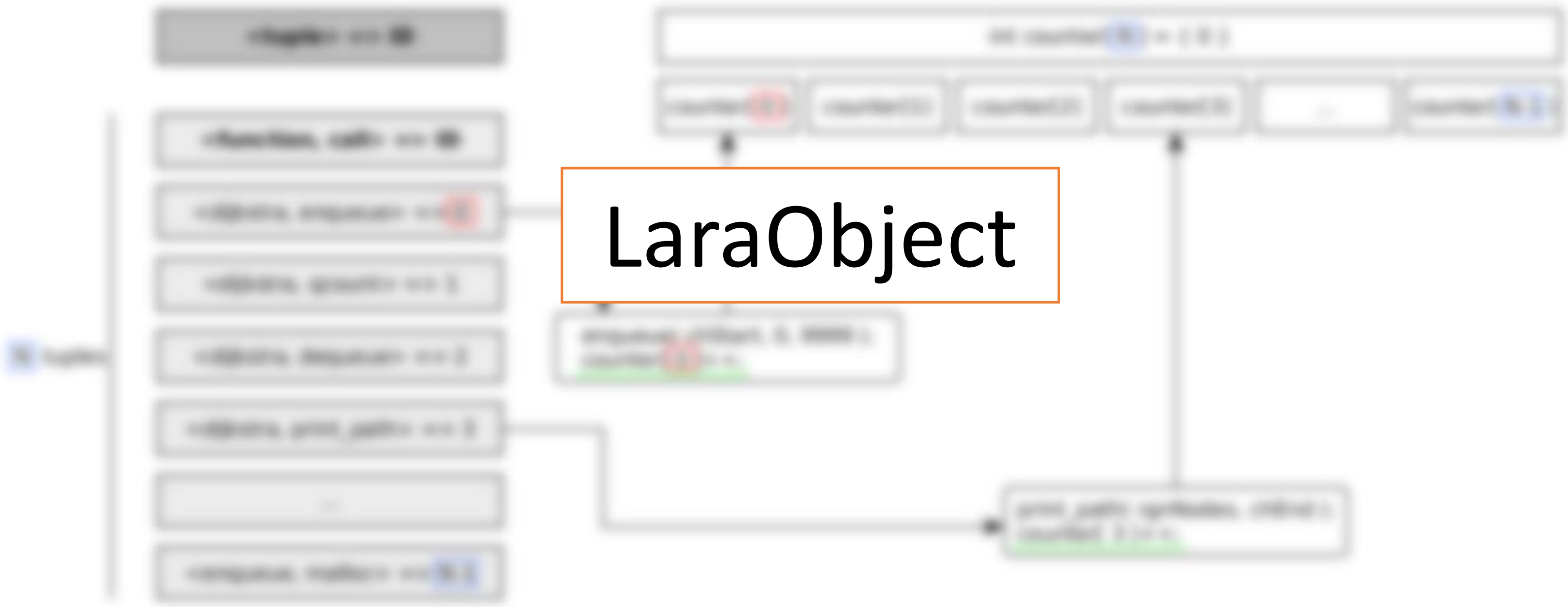
Capturing and Using Tuples



Capturing and Using Tuples



Capturing and Using Tuples



2.2 Branch Coverage – Goal

- Calculate hits counts on *if* statements
  - And corresponding *then* statements
- Possible to calculate branch frequency
- Generate a JSON report

Example output:


```
{
  "ifs": [{
      "file": "getDisparity.c",
      "line": 27,
      "total": 1,
      "positive": 1
  }, {
      "file": "findDisparity.c",
      "line": 19,
      "total": 132710400,
      "positive": 12289820
  }]
}
```

2.2 Branch Coverage – Strategy

- Select *if* statement and *then* block
  - Associate an unique id
- Declare counting arrays
  - counter[id] → *if* statement
  - pos_counter[id] → *then* block
- Inject code to increment counters
  - Before *if* → total
  - Inside *then* → positive branch
- Insert code to print JSON report

```
select file.function.if.then.first_stmt end
apply
 var id = ifIDs.getId($function.name, $if.rank);

 ifInfo.set($function.name,
            $if.rank,
            {'line': $if.line, 'file':$file.name});

 $if.insert before 'counter[[[id]]]++;';
 $first_stmt.insert before 'pos_counter[[[id]]]++;';
end
```

2.2 Branch Coverage – Diff

```c
void grid_iterate_kernel( float potential[32][64][16], int obstacles[32][64][16] )
{
    unsigned int i, j, k, steps;
    int val;
    float acc;
    float c = ( 1.0F / 6.0F );

    for ( steps = 0; steps < ( 3 * ( 64 / 16 ) ); steps ++ ) {
        for ( i = 1; i < ( 32 - 1 ); i += 1 ) {
            for ( j = 1; j < ( 64 - 1 ); j ++ ) {
                for ( k = 1; k < ( 16 - 1 ); k ++ ) {

                    val = obstacles[i][j][k];
                    if ( ( val == 1 ) ) {
                        potential[i][j][k] = 0.0F;
                    } else {
                        if ( ( val == ( - 1 ) ) ) {
                            potential[i][j][k] = 1.0F;
                        } else {
                            acc =   potential[( i - 1 )][j][k] +
                                    potential[( i + 1 )][j][k] +
                                    potential[i][( j - 1 )][k] +
                                    potential[i][( j + 1 )][k] +
                                    potential[i][j][( k - 1 )] +
                                    potential[i][j][( k + 1 )];
                            potential[i][j][k] = ( acc * c );
                        }
                    }
                }
            }
        }
    }
}
```

```c
extern int __if_counter[ 2 ];
extern int __if_pos_counter[ 2 ];

void grid_iterate_kernel( float potential[32][64][16], int obstacles[32][64][16] )
{
    unsigned int i, j, k, steps;
    int val;
    float acc;
    float c = ( 1.0F / 6.0F );

    for ( steps = 0; steps < ( 3 * ( 64 / 16 ) ); steps ++ ) {
        for ( i = 1; i < ( 32 - 1 ); i += 1 ) {
            for ( j = 1; j < ( 64 - 1 ); j ++ ) {
                for ( k = 1; k < ( 16 - 1 ); k ++ ) {

                    val = obstacles[i][j][k];
                    __if_counter[ 0 ]++;
                    if ( ( val == 1 ) ) {
                        __if_pos_counter[ 0 ]++;
                        potential[i][j][k] = 0.0F;
                    } else {
                        __if_counter[ 1 ]++;
                        if ( ( val == ( - 1 ) ) ) {
                            __if_pos_counter[ 1 ]++;
                            potential[i][j][k] = 1.0F;
                        } else {
                            acc =   potential[( i - 1 )][j][k] +
                                    potential[( i + 1 )][j][k] +
                                    potential[i][( j - 1 )][k] +
                                    potential[i][( j + 1 )][k] +
                                    potential[i][j][( k - 1 )] +
                                    potential[i][j][( k + 1 )];
                            potential[i][j][k] = ( acc * c );
                        }
                    }
                }
            }
        }
    }
}
```

Counter Declaration

Counter Increments

Main LARA code:

```
$if.insert before 'counter[[[id]]]++;';
$first_stmt.insert before 'pos_counter[[[id]]]++;';
```

2.2 Branch Coverage – Diff

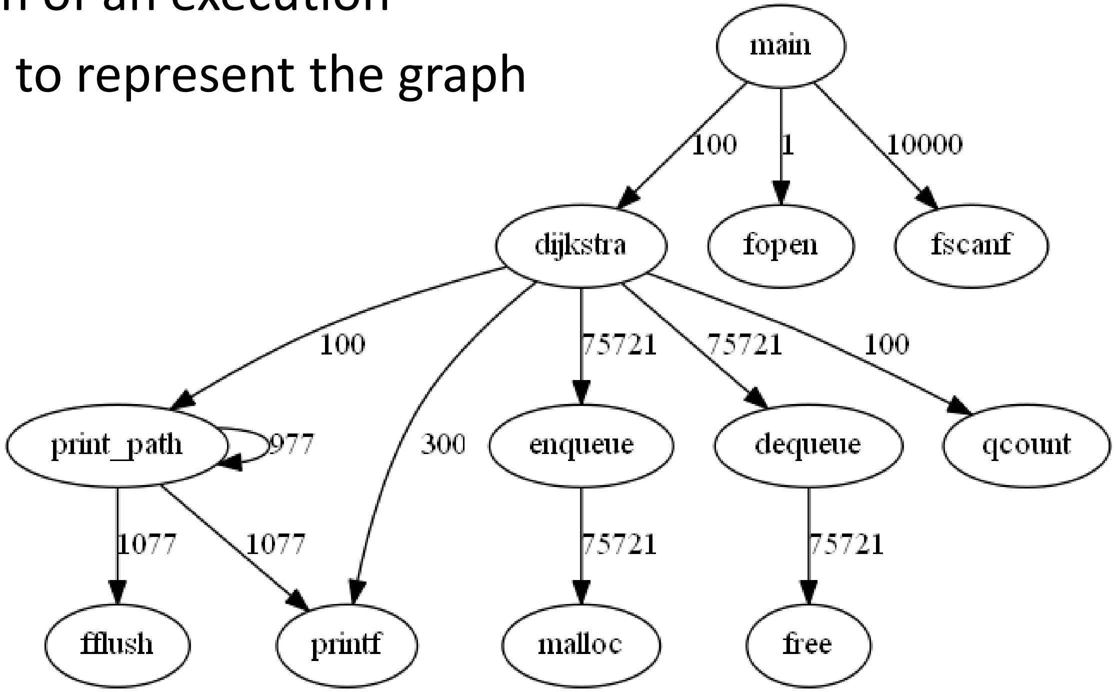
```
for ( steps = 0; steps < ( 3 * ( 64 / 16 ) ); steps ++ ) {
    for ( i = 1; i < ( 32 - 1 ); i += 1 ) {
        for ( j = 1; j < ( 64 - 1 ); j ++ ) {
            for ( k = 1; k < ( 16 - 1 ); k ++ ) {

                val = obstacles[i][j][k];
                __if_counter[ 0 ]++;
                if ( ( val == 1 ) ) {
                    __if_pos_counter[ 0 ]++;
                    potential[i][j][k] = 0.0F;
                } else {
                    __if_counter[ 1 ]++;
                    if ( ( val == ( - 1 ) ) ) {
                        __if_pos_counter[ 1 ]++;
                        potential[i][j][k] = 1.0F;
                    } else {
                        acc =   potential[( i - 1 )][j][k] +
                                potential[( i + 1 )][j][k] +
                                potential[i][( j - 1 )][k] +
                                potential[i][( j + 1 )][k] +
                                potential[i][j][( k - 1 )] +
                                potential[i][j][( k + 1 )];
                        potential[i][j][k] = ( acc * c );
                    }
                }
            }
        }
    }
}
```

Same if
=
Same ID

Same if
=
Same ID

2.3 Dynamic Call Graph (DCG) – Goals

- Capture a call graph of an execution
- Generate dot code to represent the graph



2.3 Dynamic Call Graph (DCG) – Strategy

- Select <function, call> pairs
  - Define id for each pair
  - Use id to insert counter increment before calls

- Insert counting array
  - counter[id] → pair

- Add print code to output dot

```
select function.call end
 apply
  var id = obj.getId($function.name, $call.name);
  insert after 'call_graph[ [[id]] ]++;';
 end
end
```

2.3.1 Complex DCG – Goals

- Extend previous DCG
  - Also capture execution time of call
- Generate a JSON containing timing information
- Use a second aspect to read JSON and print dot code
- This aspect can be customized and extended
  - Other approaches cannot

2.3.1 Complex DCG – Strategy

- Add timing strategy
  - *tic* before call
  - *toc* after call

```
select function.call end
apply
 var id = obj.getId($function.name, $call.name);

 var inc = cgm.inc(id);
 insert before '[[inc]]';

 var tic = cgm.tic(id);
 insert before '[[tic]]';

 var toc = cgm.toc(id);
 insert after '[[toc]]';
end
```

- Insert code to print JSON report

- A second LARA aspect to read report and print dot

2.3.1 Complex DCG – Tic and Toc Functions

```
void tic(int id) {

 gettimeofday( & cgm_tmp_timers[id], NULL);
}
```

```
void toc(int id) {

 struct timeval end;
 unsigned long seconds, useconds;

 gettimeofday( & end, NULL);
 seconds = end.tv_sec - cgm_tmp_timers[id].tv_sec;
 useconds = end.tv_usec - cgm_tmp_timers[id].tv_usec;

 cgm_timers[id] += seconds * 1000000 + useconds;
}
```

2.3.1 Complex DCG – Diff

```c
#include <stdio.h>
#include <stdlib.h>
#include "disparity.h"

void correlateSAD_2D( I2D * Ileft, I2D * Iright, I2D * Iright_moved,
                      int win_sz, int disparity, F2D * SAD, F2D * integralImg, F2D * retSAD )
{
    int rows, cols;
    int i, j, endRM;
    I2D * range;

    range = iMallocHandle( 1, 2 );

    range->data[( ( 0 * range->width ) + 0 )] = 0;
    range->data[( ( 0 * range->width ) + 1 )] = disparity;
    rows = Iright_moved->height;
    cols = Iright_moved->width;
    for ( i = 0; i < ( rows * cols ); i ++ ) {
        Iright_moved->data[i] = 0;
    }

    padarray4( Iright, range, ( - 1 ), Iright_moved );

    computeSAD( Ileft, Iright_moved, SAD );

    integralImage2D2D( SAD, integralImg );

    finalSAD( integralImg, win_sz, retSAD );

    return ;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "disparity.h"
#include <sys/time.h>
struct timeval cgm_tmp_timers[21];
unsigned long  cgm_timers[21];
unsigned int cgm_counters[21];
void tic( int id );
void toc( int id );

void correlateSAD_2D( I2D * Ileft, I2D * Iright, I2D * Iright_moved,
                      int win_sz, int disparity, F2D * SAD, F2D * integralImg, F2D * retSAD )
{
    int rows, cols;
    int i, j, endRM;
    I2D * range;

    cgm_counters[10]++;
    tic( 10 );
    range = iMallocHandle( 1, 2 );
    toc( 10 );

    range->data[( ( 0 * range->width ) + 0 )] = 0;
    range->data[( ( 0 * range->width ) + 1 )] = disparity;
    rows = Iright_moved->height;
    cols = Iright_moved->width;
    for ( i = 0; i < ( rows * cols ); i ++ ) {
        Iright_moved->data[i] = 0;
    }

    cgm_counters[11]++;
    tic( 11 );
    padarray4( Iright, range, ( - 1 ), Iright_moved );
    toc( 11 );

    cgm_counters[12]++;
    tic( 12 );
    computeSAD( Ileft, Iright_moved, SAD );
    toc( 12 );

    cgm_counters[13]++;
    tic( 13 );
    integralImage2D2D( SAD, integralImg );
    toc( 13 );

    cgm_counters[14]++;
    tic( 14 );
    finalSAD( integralImg, win_sz, retSAD );
    toc( 14 );

    return ;
}
```

Main LARA code:

```
insert before '[[inc]]';

insert before '[[tic]]';
insert after '[[toc]]';
```

2.3.1 Complex DCG – Diff

```c
#include <stdio.h>
#include <stdlib.h>
#include "disparity.h"

void correlateSAD_2D( I2D * Ileft, I2D * Iright, I2D * Iright_moved,
                      int win_sz, int disparity, F2D * SAD, F2D * integralImg, F2D * retSAD )
{
    int rows, cols;
    int i, j, endRM;
    I2D * range;

    range = iMallocHandle( 1, 2 );

    range->data[( ( 0 * range->width ) + 0 )] = 0;
    range->data[( ( 0 * range->width ) + 1 )] = disparity;
```

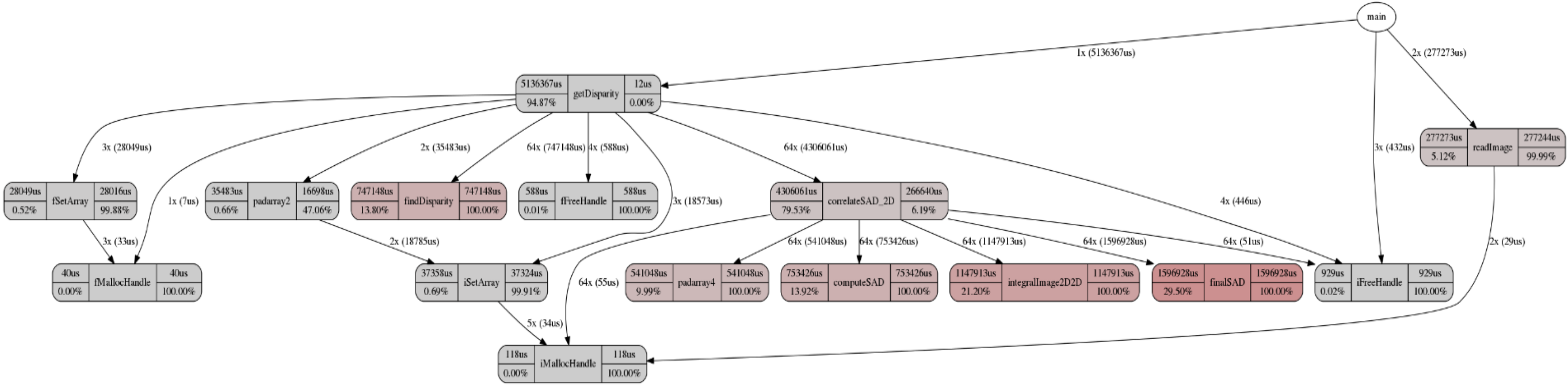
```c
#include <stdio.h>
#include <stdlib.h>
#include "disparity.h"
#include <sys/time.h>
struct timeval cgm_tmp_timers[21];
unsigned long  cgm_timers[21];
unsigned int cgm_counters[21];
void tic( int id );
void toc( int id );

void correlateSAD_2D( I2D * Ileft, I2D * Iright, I2D * Iright_moved,
                        int win_sz, int disparity, F2D * SAD, F2D * integralImg, F2D * retSAD )
{
    int rows, cols;
    int i, j, endRM;
```

2.3.1 Complex DCG – Diff

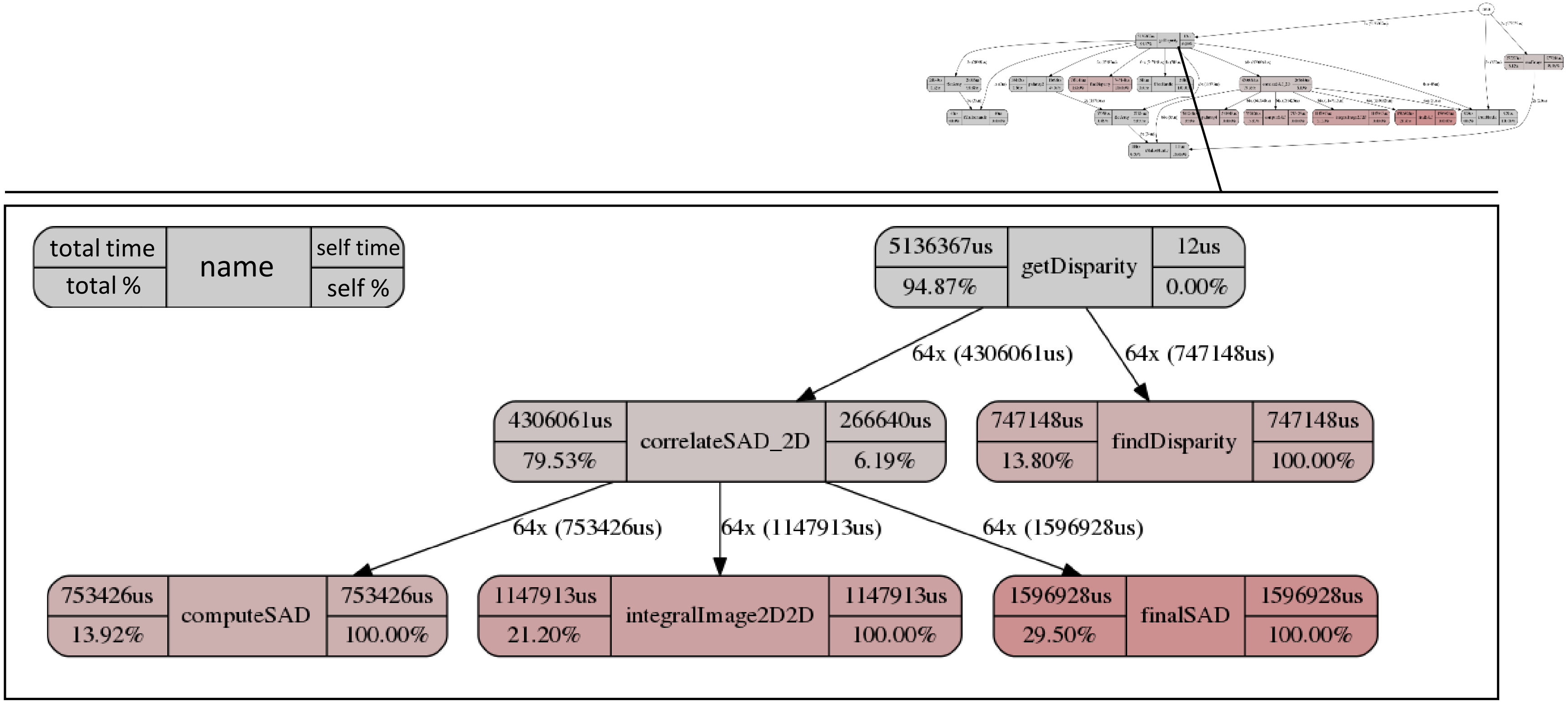
```
I2D * range;

range = iMallocHandle( 1, 2 );

range->data[( ( 0 * range->width ) + 0 )] = 0;
range->data[( ( 0 * range->width ) + 1 )] = disparity;
rows = Iright_moved->height;
cols = Iright_moved->width;
for ( i = 0; i < ( rows * cols ); i ++ ) {
    Iright_moved->data[i] = 0;
}

padarray4( Iright, range, ( - 1 ), Iright_moved );
```

```
void correlateSAD_2D( I2D * Ileft, I2D * Iright, I2D * Iright_moved,
                      int win_sz, int disparity, F2D * SAD, F2D * integralImg, F2D * retSAD )
{
    int rows, cols;
    int i, j, endRM;
    I2D * range;

    cgm_counters[10]++;
    tic( 10 );
    range = iMallocHandle( 1, 2 );
    toc( 10 );

    range->data[( ( 0 * range->width ) + 0 )] = 0;
```

2.3.1 Complex DCG – Example Graph



2.3.1 Complex DCG – Example Graph



2.4 Range Values Monitoring – Goal

- Collect range of variables for a given function
- Report the ranges

Expected output:

```
computeSAD
 rows: {296.000000, 296.000000}
 cols: {360.000000, 360.000000}
 diff: {-190.000000, 221.000000}
 data_temp_0: {0.000000, 48841.000000}
```

2.4 Range Values Monitoring – Strategy

- Deal with possible variables inside structs
  - Using a support transformation

```
aspectdef PrepareStructs
  input funcName end

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

- Insert supporting code
  - Arrays to store values, indexed by ID
  - Initialize, update and printing functions

2.4 Range Values Monitoring – Strategy

- Select variables inside a target function
  - Generate ID using function and variable names
  - Update range after a variable is assigned
  - Filter by write and location

```
select function{funcName}.var end
apply
  var id = vars.getId($function.name, $var.name);
  insert after 'range_update([[id]], [[$var.name]]);';
end
condition
  !$var.in_loop_header &&
   $var.reference == 'write' &&
  !$var.is_struct
end
```

2.4 Range Values Monitoring – Diff

```c
#include <stdio.h>
#include <stdlib.h>
#include "disparity.h"

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;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "disparity.h"

void range_update( unsigned int id, double value );
extern double range_min[ 4 ];
extern double range_max[ 4 ];
void computeSAD(I2D *Ileft, I2D* Iright_moved, F2D* SAD)
{
    int rows, cols, i, j, diff;

    rows = Ileft->height;
    range_update( 0, rows );
    cols = Ileft->width;
    range_update( 1, cols );

    for(i=0; i<rows; i++) {
        for(j=0; j<cols; j++) {
            float data_temp_0;
            diff =  Ileft->data[( ( i * Ileft->width ) + j )] -
                    Iright_moved->data[( ( i * Iright_moved->width ) + j )];
            range_update( 2, diff );

            data_temp_0 = ( diff * diff );
            range_update( 3, data_temp_0 );

            SAD->data[( ( i * SAD->width ) + j )] = data_temp_0;
        }
    }

    return;
```

Main LARA code:

```
insert after 'range_update([[id]], [[$var.name]]);';
!$var.in_loop_header && $var.reference == 'write' && !$var.is_struct
```

Takeaway Points

- Different dynamic analysis, all based on instrumentation

- *insert* action
  - A simple but powerful/versatile transformation
  - Instrumentation mechanism for monitoring, profiling, debugging,...

- LARA enables easy selection of interesting code points
  - Join point attributes to filter results