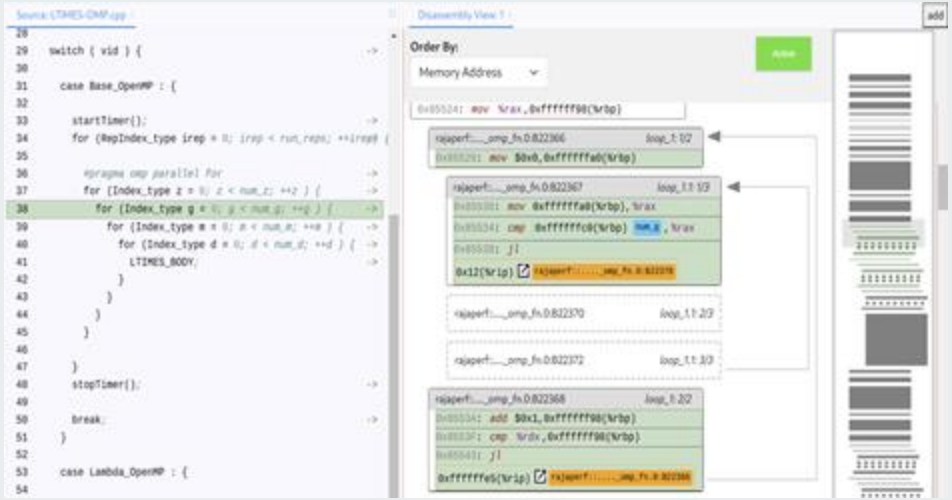
Interactive Visualization of Binary Code for Investigating Compiler Optimizations



Shadmaan Hye
Graduate Student, University of Utah

June 19, 2023

Problem: Labor Intensive Compilation Analys

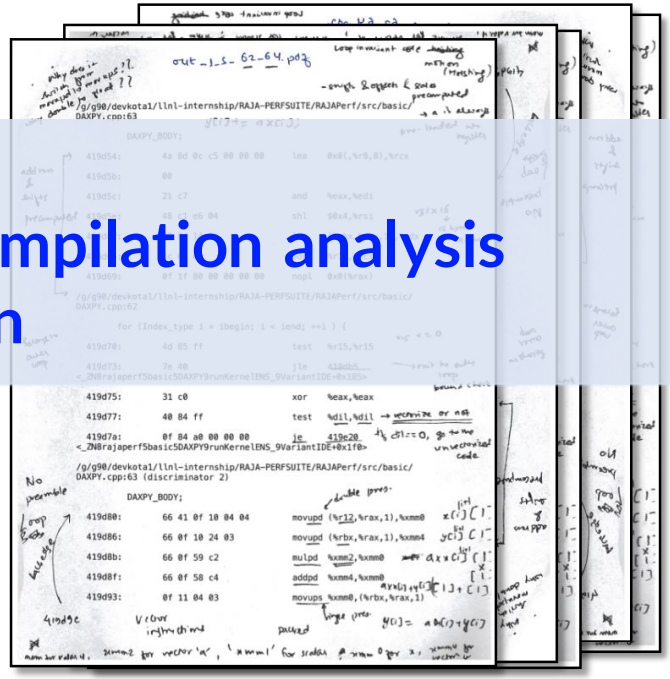
Multiple Compilers (gcc, intel, clang, ..)

Multiple Compilation Options (-O2, -O3, -funroll-loops, ..)

Multiple Optimizations (inlining, vectorization, loop unrolling, variable hoisting, ..)

Enable Code Developers to understand what optimization compiler is doing to the code through a visualization system


```
#include "DAXPY.hpp"

#include "RAJA/RAJA.hpp"

#include <iostream>

namespace rajaperf
{
namespace basic
{

void DAXPY::runSeqVariant(VariantID vid)
{
  const Index_type run_reps = getRunReps();
  const Index_type ibegin = 0;
  const Index_type iend = getRunSize();

  DAXPY_DATA_SETUP;

  auto daxpy_lam = [=](Index_type i) {
                     DAXPY_BODY;
                   };

  switch ( vid ) {

    case Base_Seq : {
```



Main Goal

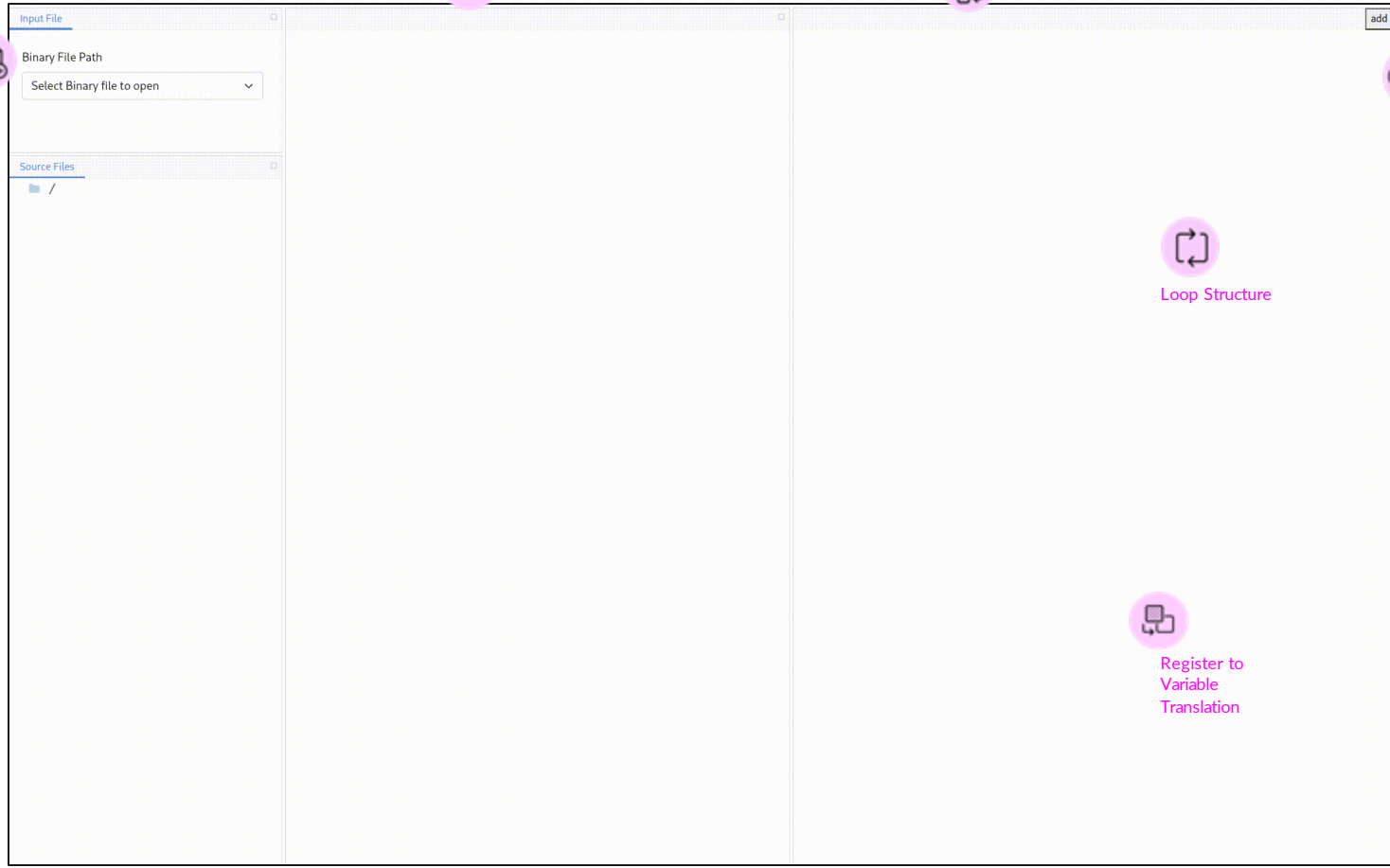
To improve the human side of compilation analysis with visualization

Our Visualization Interface

Source code shown in Source: [file.name] tab

Disassembly code shown in order

Take binary file compiled with debug flag as input



Disassembly code and current location overview

File structures shown here

Loop Structure

Register to Variable Translation

Obtaining Disassembly Info

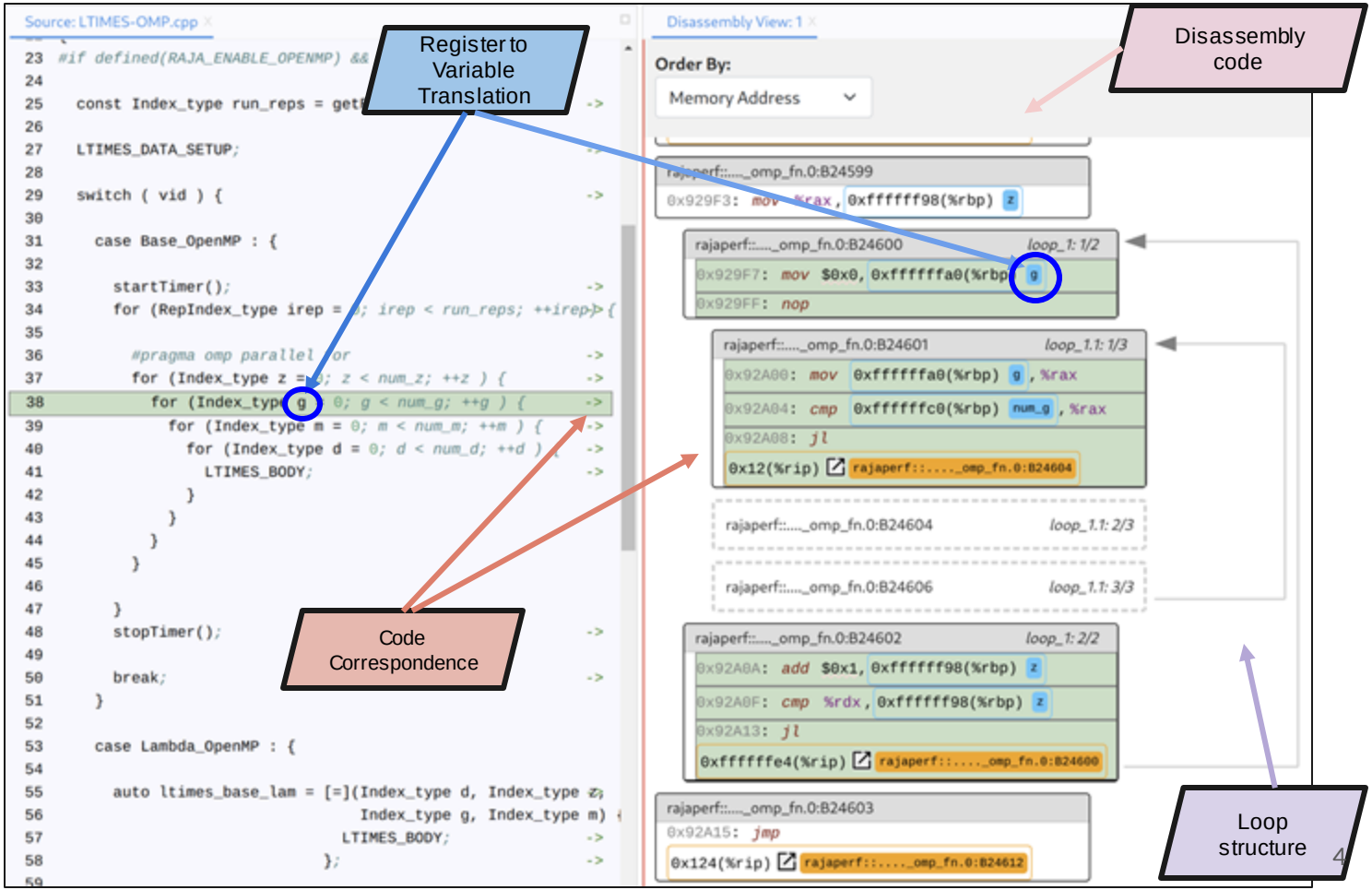
Binary Input

Debug flag

Dyninst

Disassemble

A Binary Analysis and Modification Framework
Generates info about the binary
Access to local (stack) variables



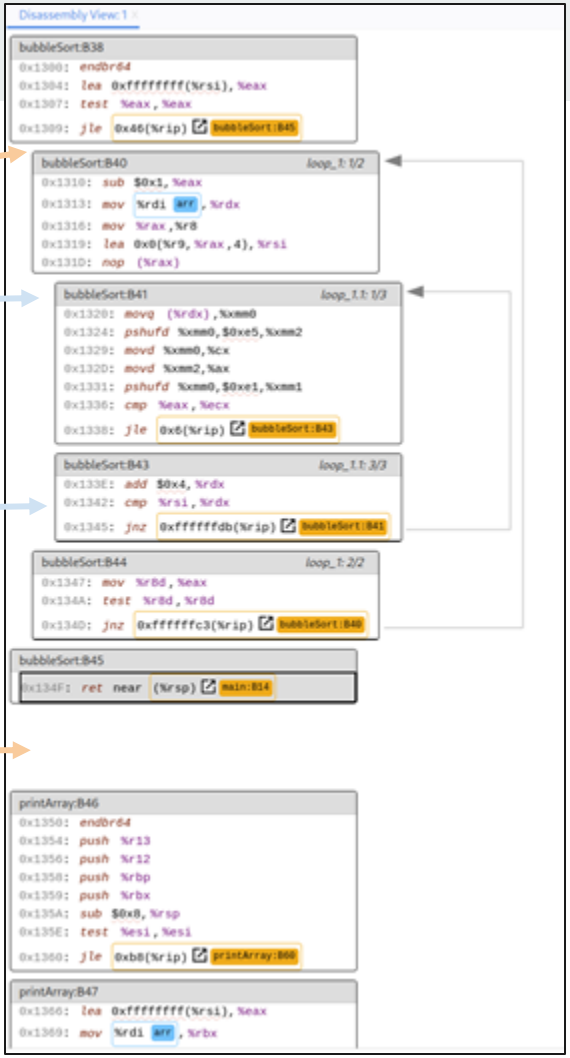
Disassembly View

No vertical space between blocks of one function

Block Header:
Function_name: B<Unique id>

Block Body:
Address: **operator operand, operand**

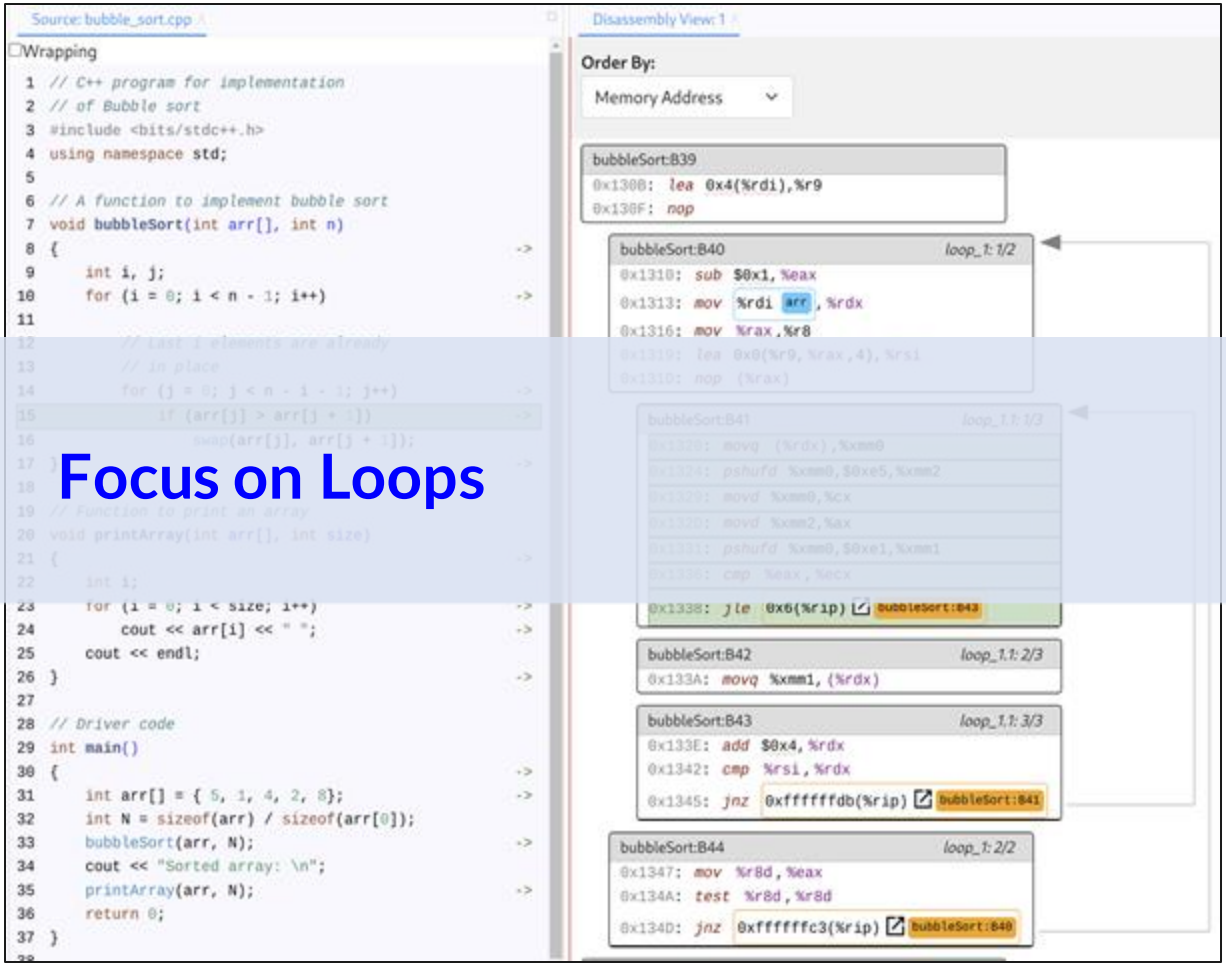
Vertical space between blocks of different function



Focus on Loops for Optimization

Programs spend the most time executing loops

Understanding which instructions are executing multiple times is vital



Focus on Loops

Ideal Loop Constructions

Blocks ordered by Virtual Memory Address

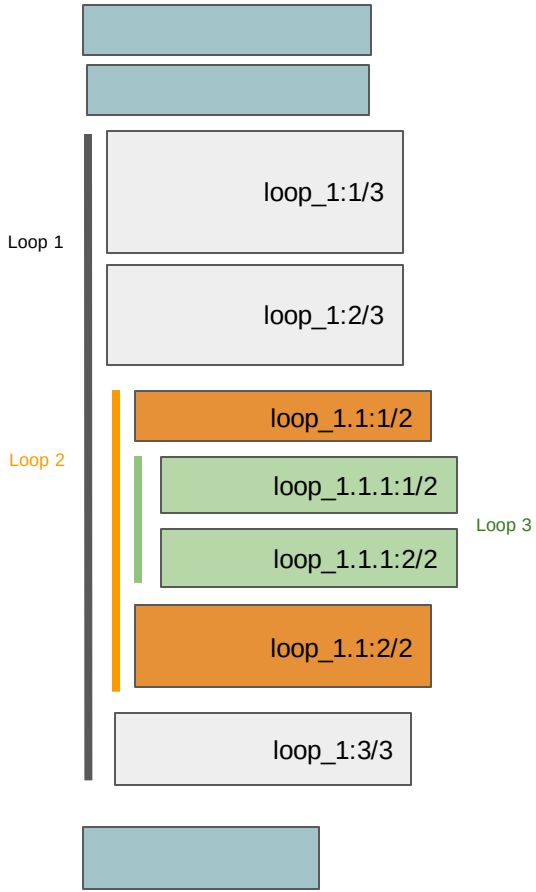
- **Indentation:** Nested Loop blocks
- **Numbering loop blocks:**
  Eg. Block with indication loop_1.1.1:2/2

loop_1.1.1:2/2

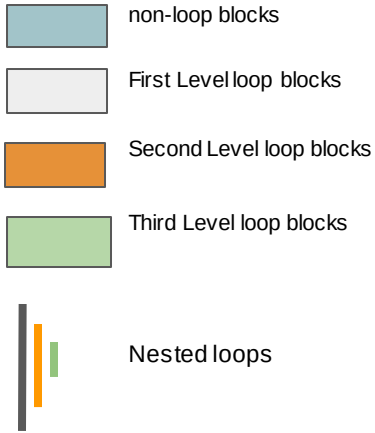
1st loop of the function

3rd level loop

2nd block out of 2 total blocks in the particular loop



Colors used for clarity



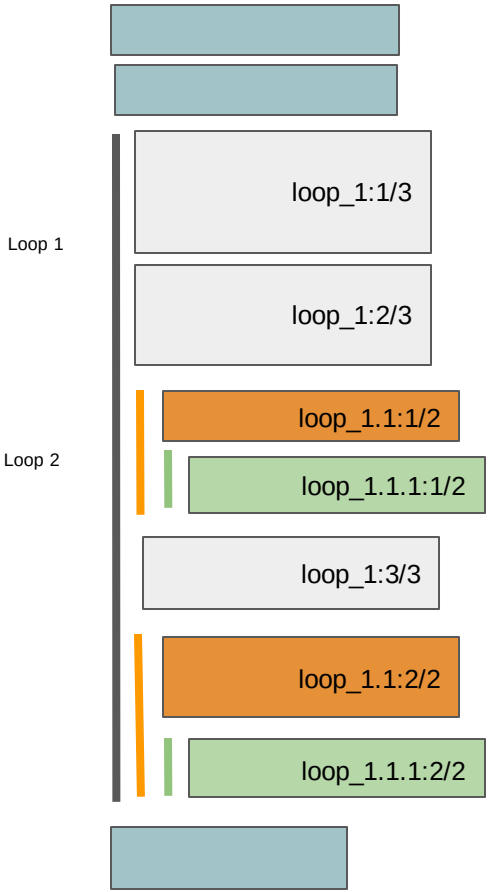
Real Case for Loops

Blocks ordered by Virtual Memory Address

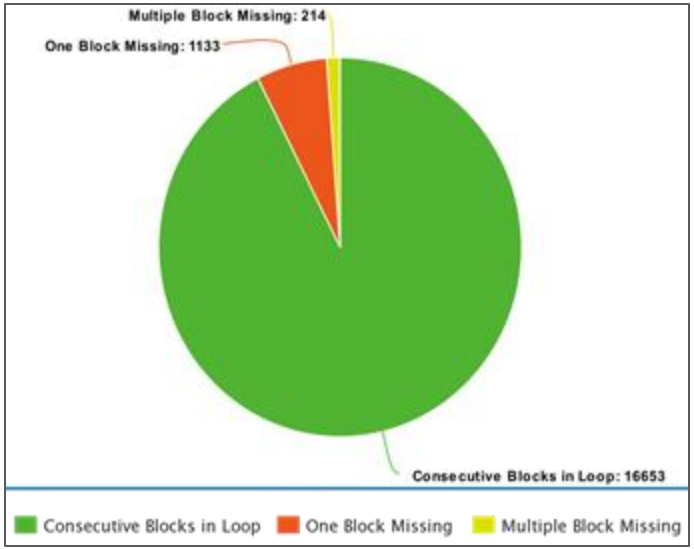
Blocks in a particular loop are not ordered sequentially

loop_1.1.1:2/2

- 1st loop of the function
- 3rd level loop
- 2nd block out of 2 total blocks in the particular loop



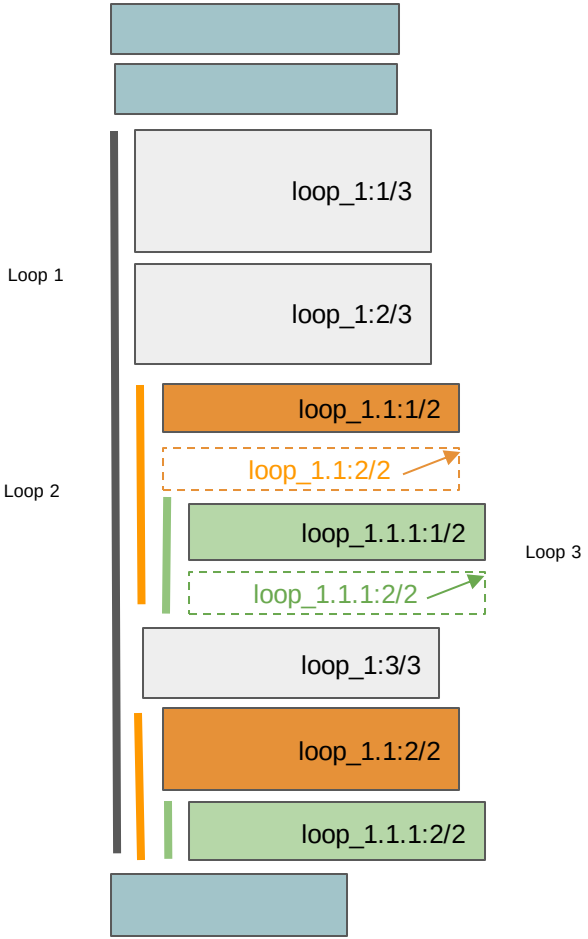
RAJA Performance Suite

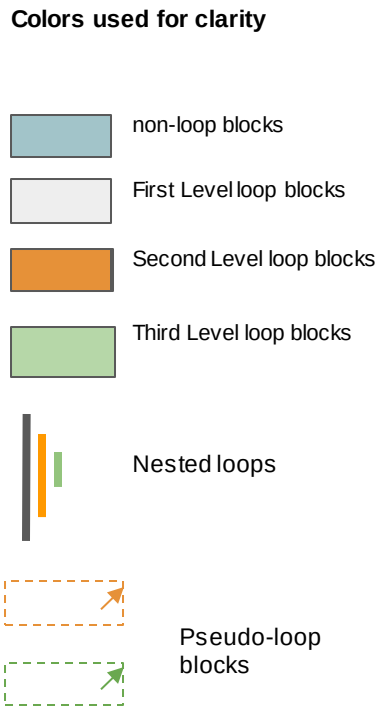


Pseudo Loop Blocks Proposal

Loop blocks have different orders.

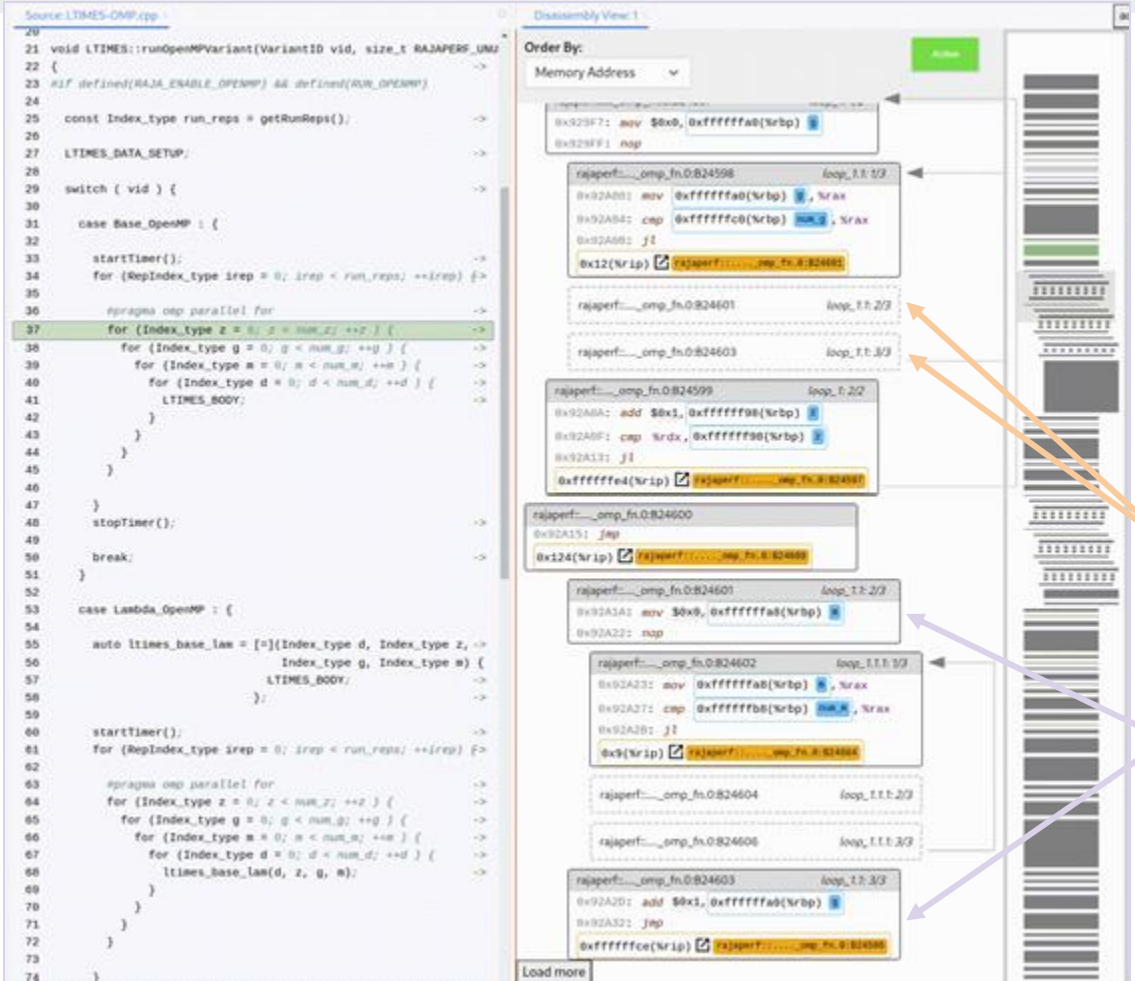
- Hard to understand which blocks are in single loop.
- To **avoid confusion** **Pseudo-loop blocks** are introduced.
- Dotted blocks arrange the instruction blocks of a particular loop together.





Pseudo Loop Blocks

In Memory Address order, the pseudo loop blocks are inserted to maintain logical order of blocks inside a loop



When pseudo blocks are clicked

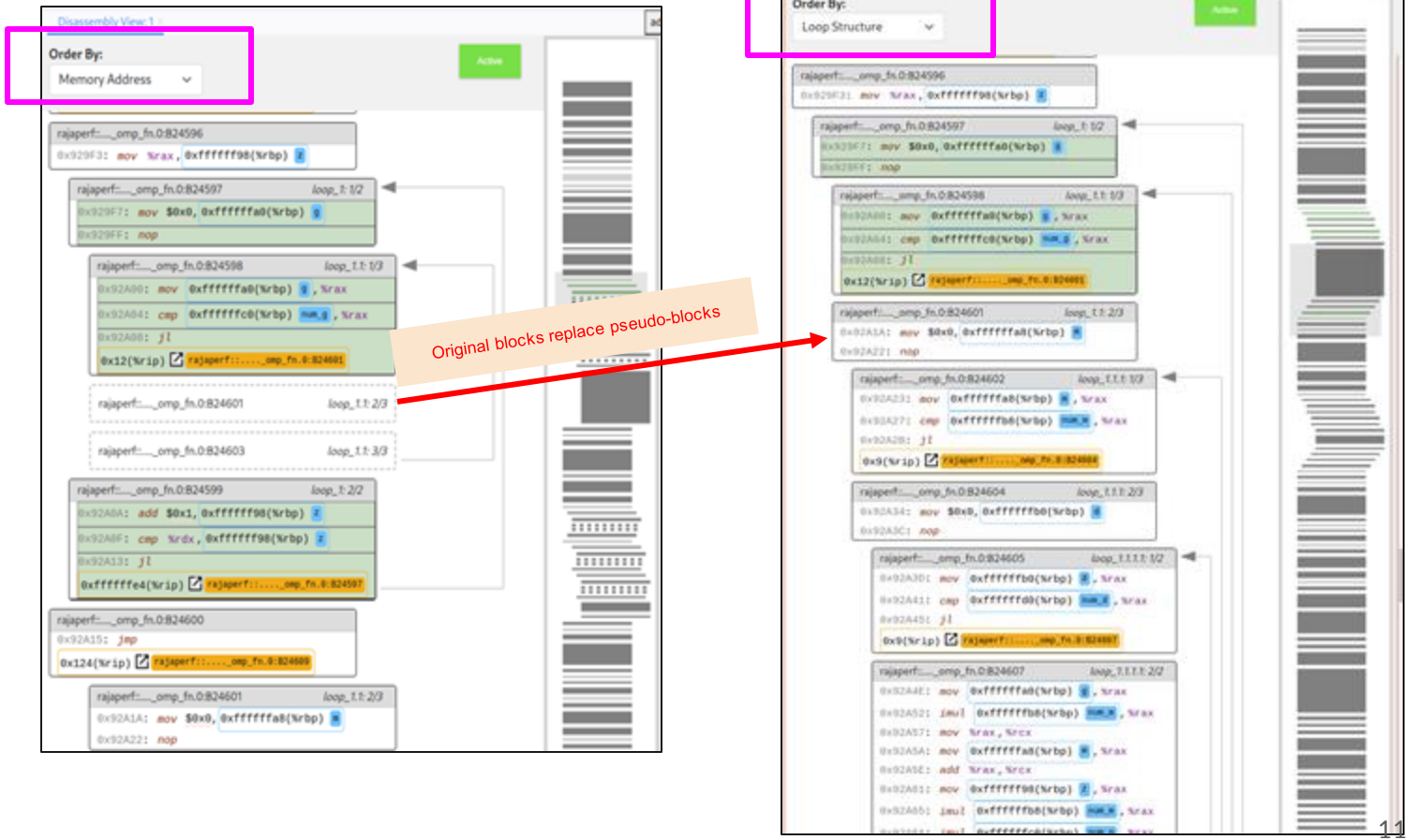
Highlights the original block

Order of Disassembly View

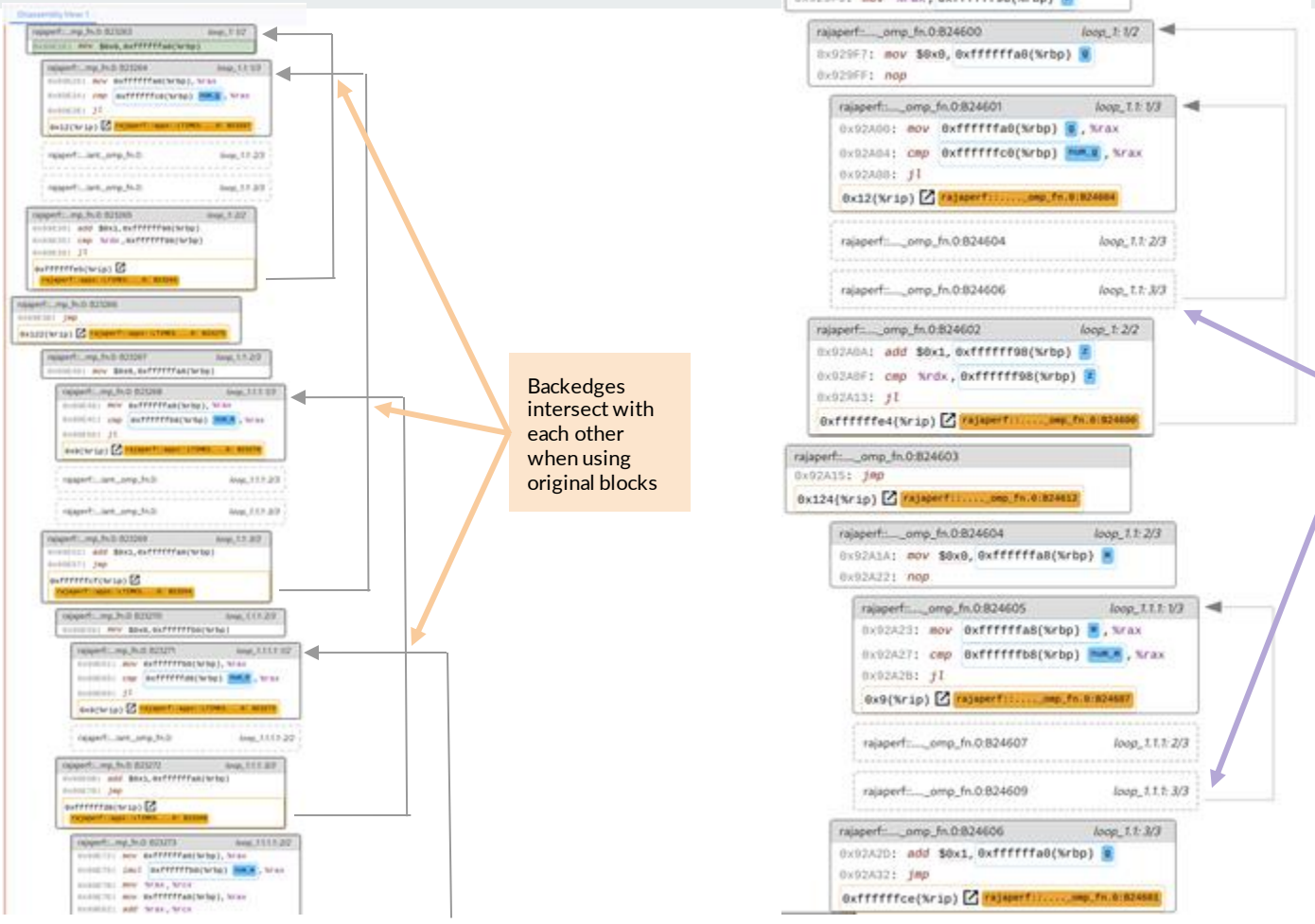
Loop blocks visualized in **2 approaches:**

1. **Memory Address**
   Order of instructions according to memory address

2. **Loop Structure**
   - Order of instructions according to loop structure.
   - Reordering blocks out of address order



Loop Backedges

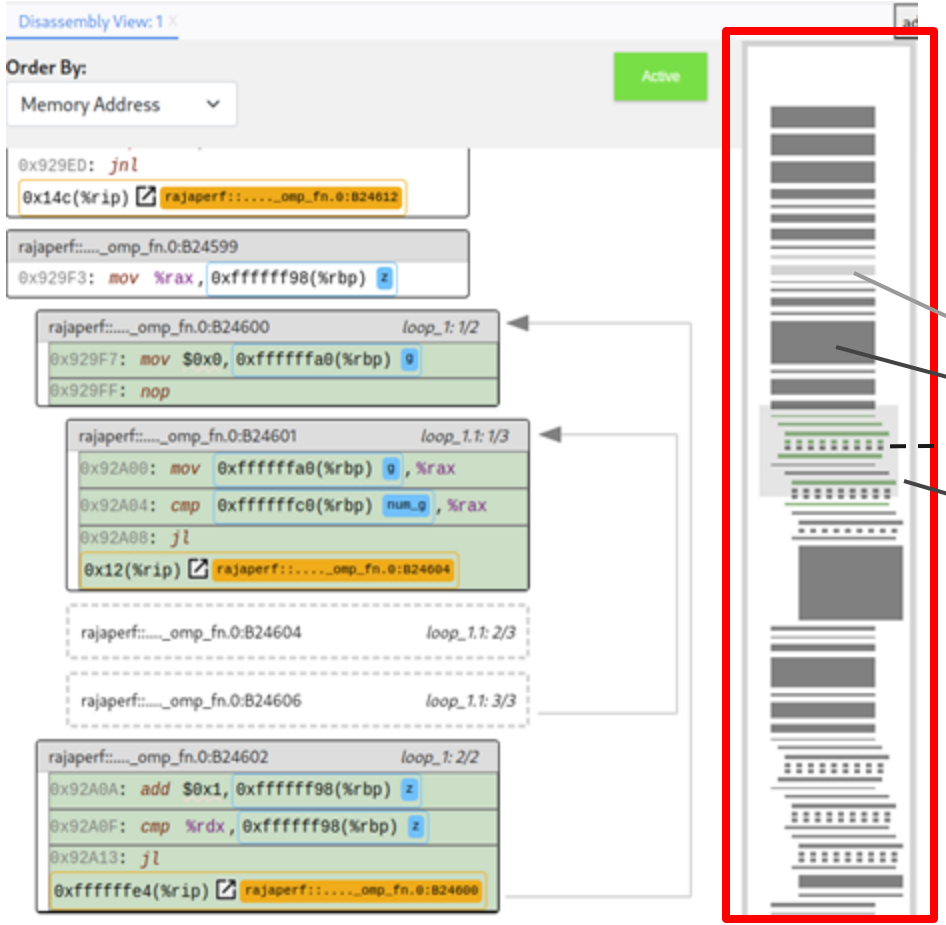


Becomes difficult to visualize at a glance

Backedges intersect with each other when using original blocks

To avoid difficulty, pseudo loop blocks used instead of original blocks.

Minimap Overview



Minimap created to get the disassembly overview

Each rectangle height - no. of instructions in blocks

Light Grey Lines - System defined functions

Dark Grey Lines - User defined functions

Dotted Lines- Pseudo loop Blocks

Indentation- Nested loops

Brush- Current Location

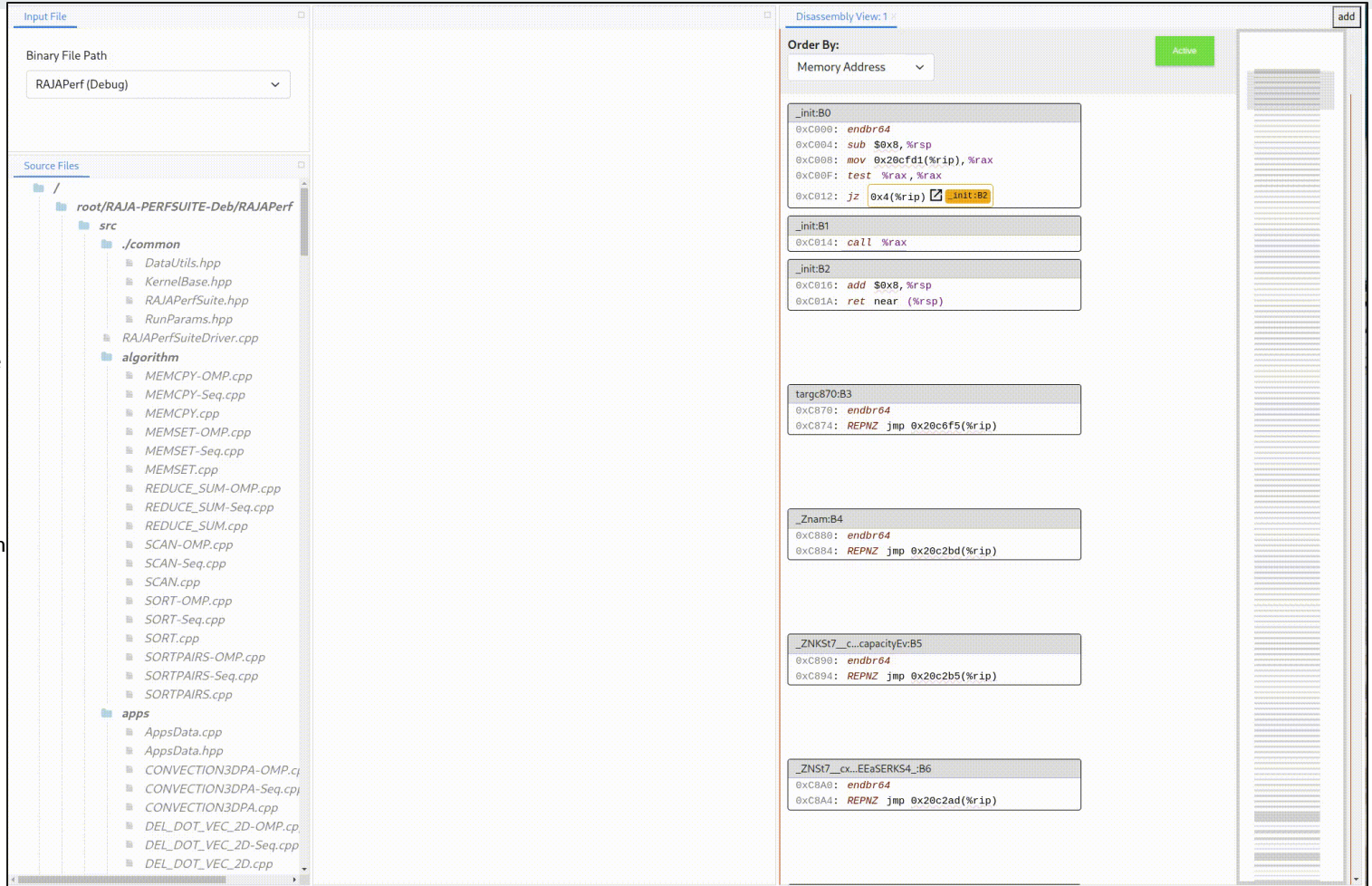
Multiple Disassembly Views

Multiple disassembly views with different colors

For visualizing multiple source code at once

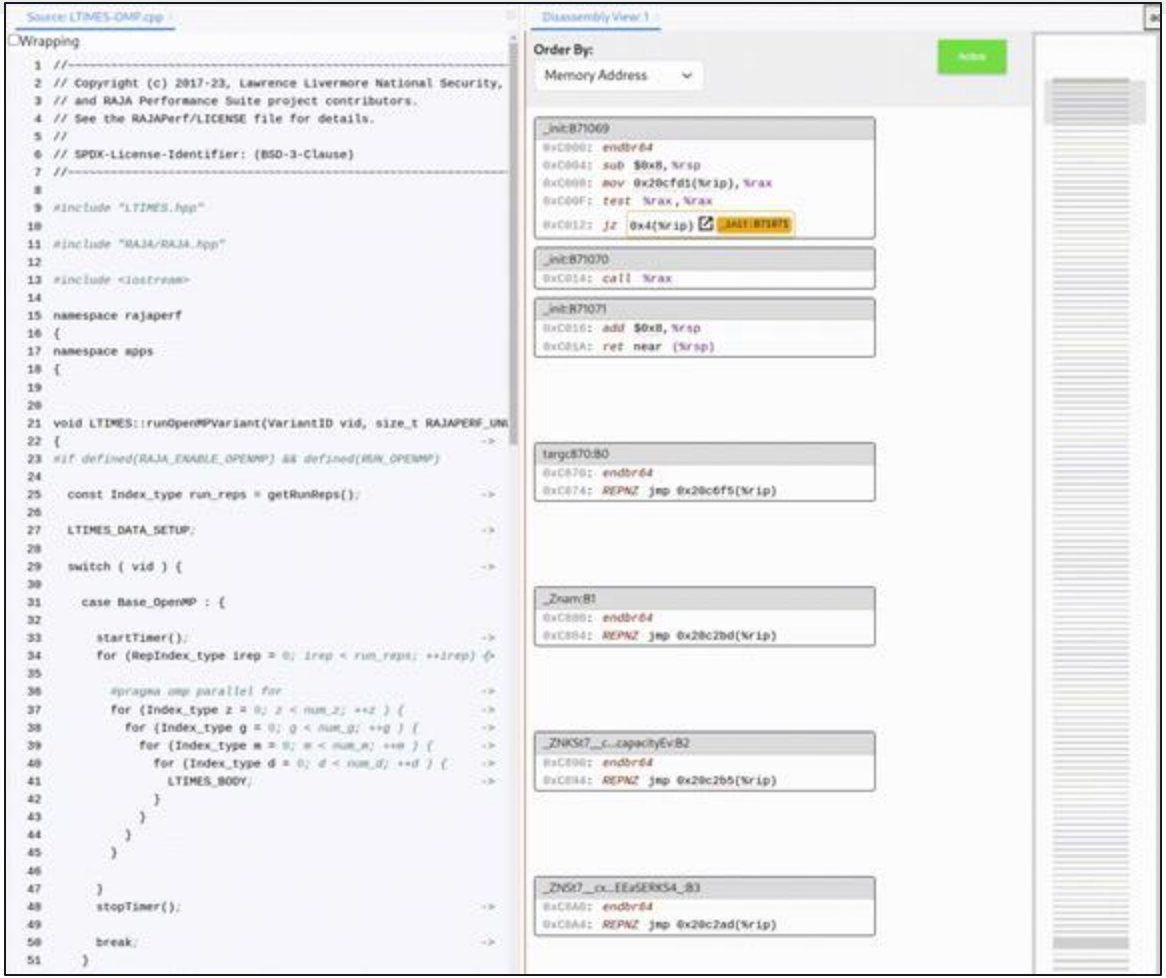
For example-

Dis View 1 - Green
Dis View 2 - Yellow
Dis View 3 - Orange and so on



Bidirectional & Many to many relation

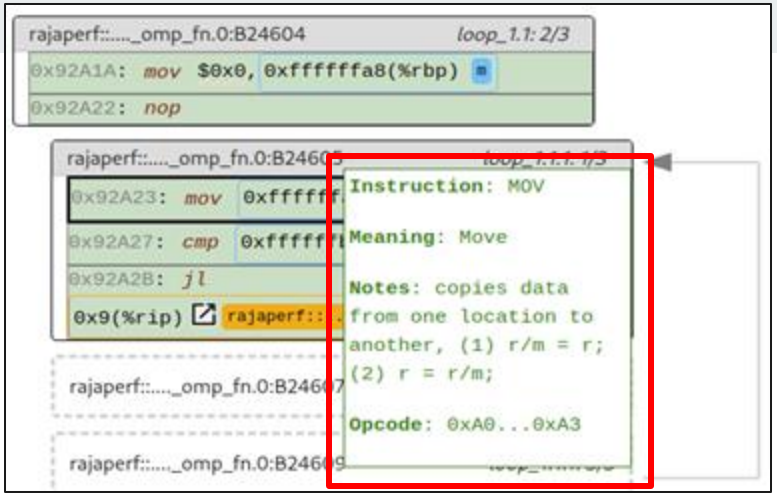
In addition with Dyninst, CcNav corresponds source and disassembly code in both ways.

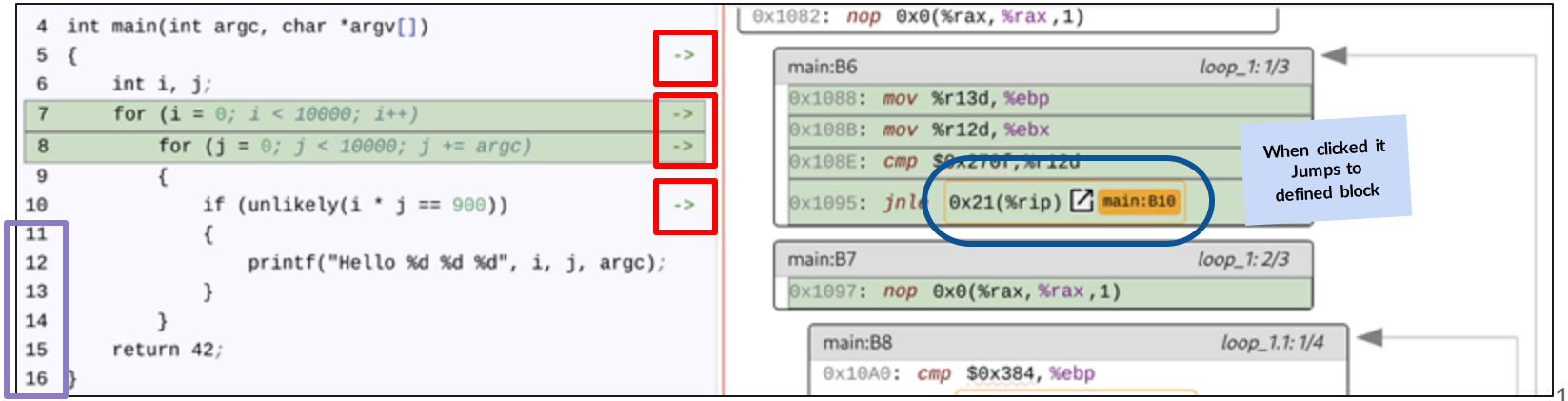


Other features

Description of the operands are shown in the tool tip

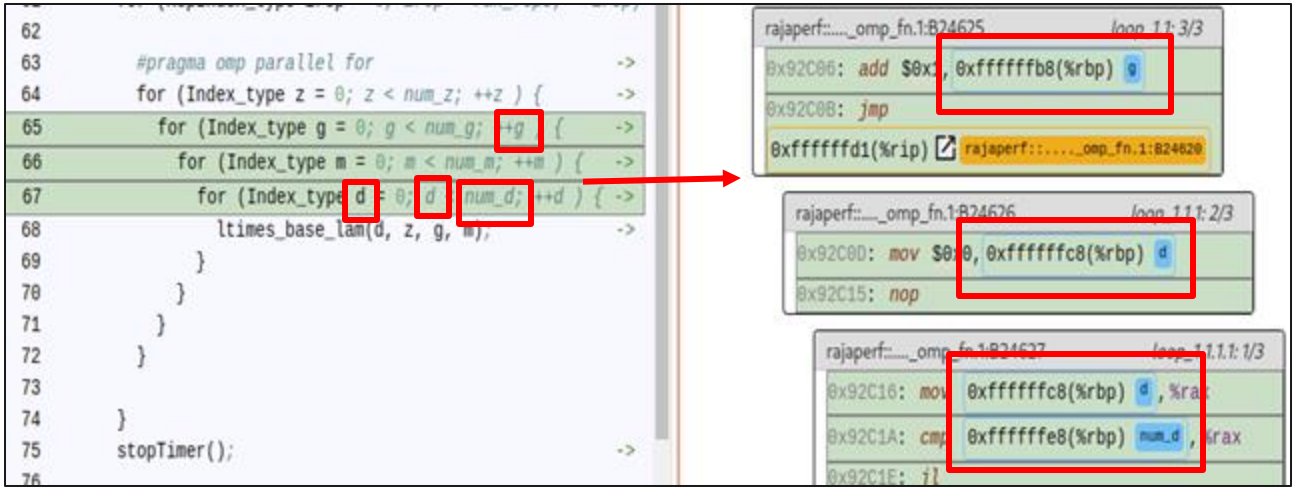
Arrows used to represent which lines have correspondence

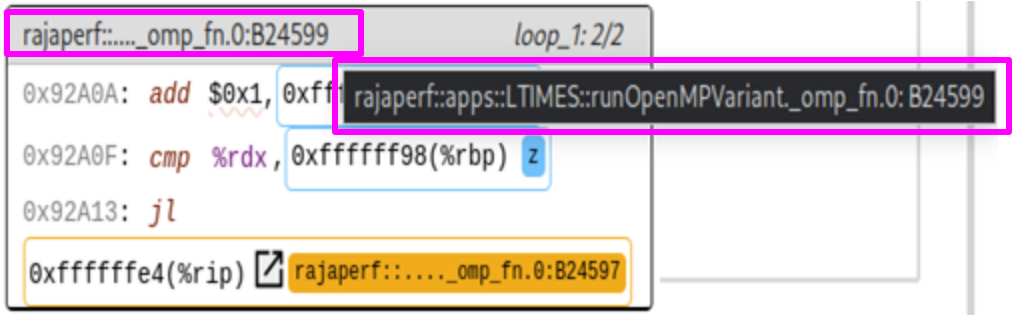




Other features (continued)

Dyninst gives information regarding variables which are renamed in the disassembly code

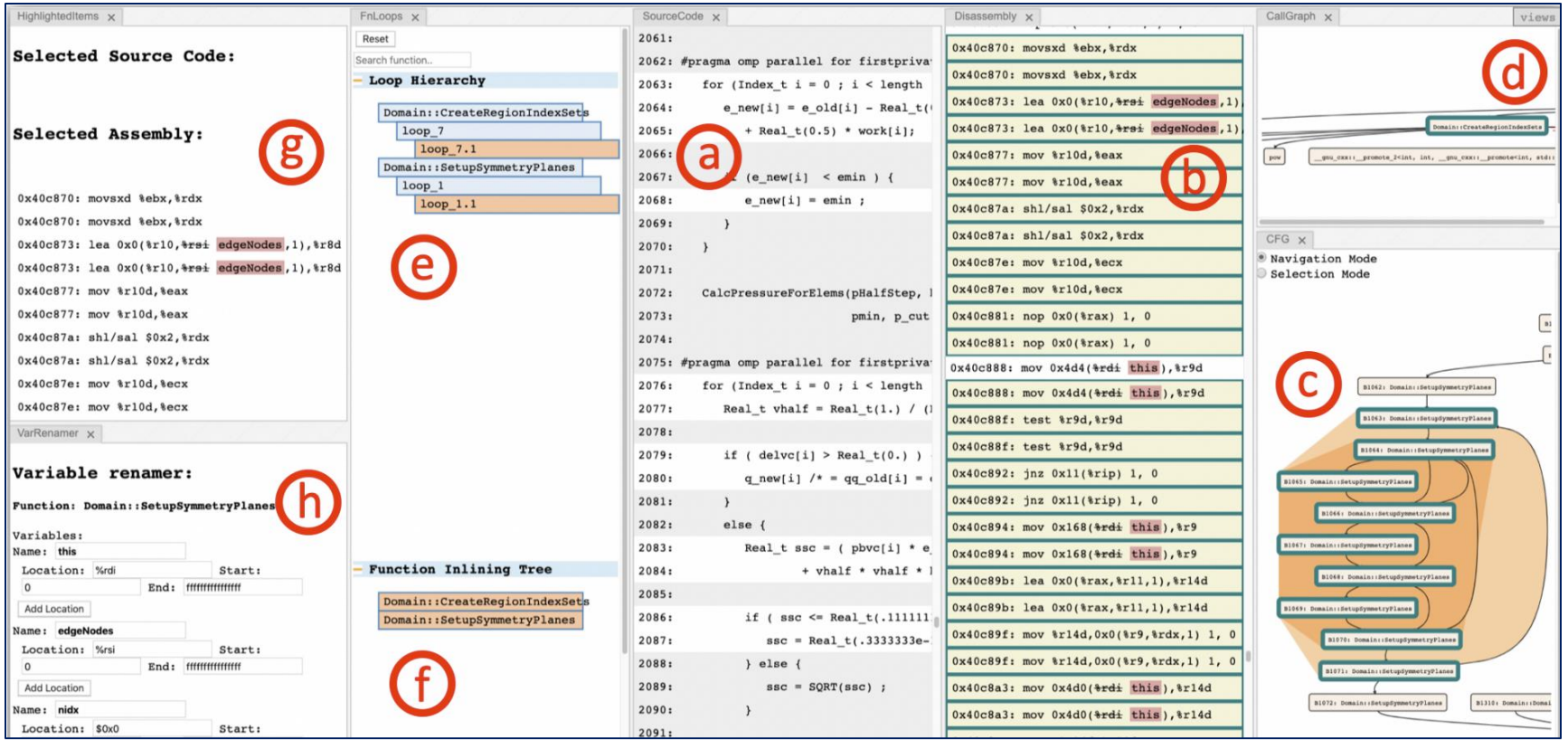




The function names are also truncated to a fixed length

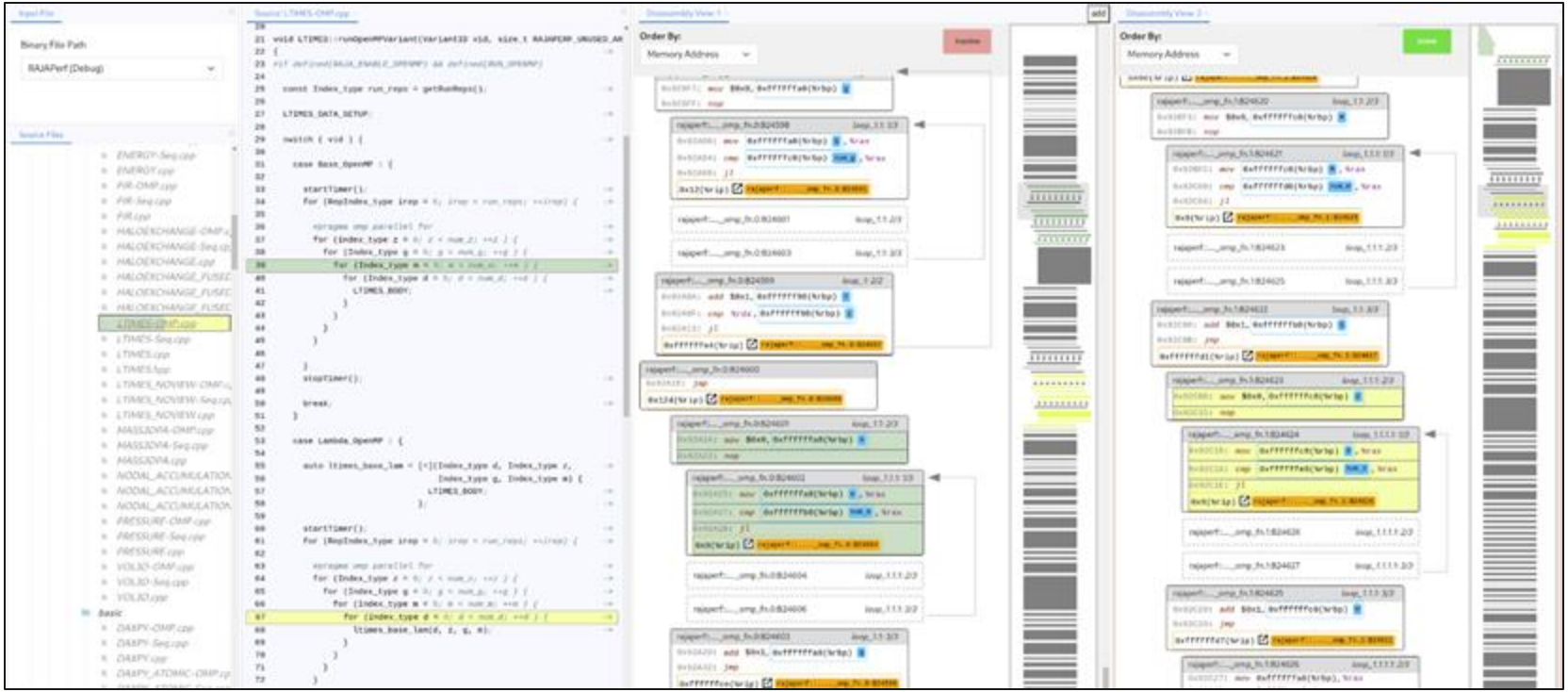
Original CcNav

The new features focus more on source & disassembly code matching
Integrate into a larger project



Thank you!

Main Goal: To improve the human side of compilation analysis with visualization



Name: Shadmaan Hye
Email: praptishadmaan@gmail.com
Github: https://github.com/Prapti-044/dis-viz.git