Interactive Visualization of Binary Code for **Compiler Optimizations**

Shadmaan Hye and Katherine E. Isaacs

Input File	Source	ELTIMES-OMP.cpp
Binary File Path	t 46	View
Dilla	48	<pre>stonTimer();</pre>
Rajaperf-Debug-00-g 🗸	49	
	50	break;
	51	}
Source	52	
Jource	53	case Lambda_OpenMP : {
Code Tree	54	
Source Files	55	<pre>auto ltimes_base_lam = [=](Index_type d, Index_</pre>
DIFFUSION3DPA-OMP.	56	Index_type g, Index_
DIFFUSION3DPA-Sea c	57	LTIMES_BODY;
	58	};
DIFFUSION3DPA.cpp	59	
DIFFUSION3DPA.hpp	60	startTimer();
EDGE3D-OMP.cpp	61	<pre>for (RepIndex_type irep = 0; irep < run_reps; +</pre>
EDGE3D-Sea.cpp	62	
= EDGE20 coo	63	#pragma omp parallel for
= EDGESD.cpp	64	for (Index_type $z = 0$; $z < num_z$; ++z) {
EDGE3D.hpp	65	for (Index_type $g = \Theta$; $g < num_g$; ++g) {
ENERGY-OMP.cpp	66	<pre>for (Index_type m = 0; m < num_m; ++m) {</pre>
ENERGY-Seq.cpp	67	<pre>for (Index_type d = 0; d < num_d; ++d)</pre>
ENERGY.cpp	68	<pre>ltimes_base_lam(d, z, g, m);</pre>
EIR-OMP con	69	}
= rin-conr.cpp	70	}
and the second sec		

By: Jump to: 📥	View tractive
ory Address v 0x0	Jump
rf: omp. fn.1:825954	
A91: mov %rax, 0xffffffb0(%rbp)) 🗷
aperf::omp_fn.1:B25955 (loop_)	header) loop_1: 1/2
9EA95: MOV \$0x0, 0xfffffb8(%rb	bp) 🤨
9EA9D: nop	
rajaperf::omp_fn.1:B25956 (loop	p_header) loop_1.1: 1/3
Bx9EA9E: mov Bxfffffb8(%rbp)	g,%rax
8x9EAA2: cmp 0xfffffd8(%rbp)	num_0,%rax
Bx9EAA6: jl rajaperf::omp_f	fn.1:825959
rajaperf:omp_fn.1:B25959	loop_1.1: 2/3
rajaperf::omp_fn.1:B25961	loop_1.1: 3/3

add	Disassembly Vi
	Order By:
	Loop Structur
	rajaperf:om
	rajaperf: 0x9EA95: 0x9EA9D: rajaperf: 0x9EA9E
	0x9EAA2 0x9EAA2 rajape rajape
	rajaperf: 9x9EAB 0x9EAB rajaperf:

	+
sassembly View. 2	SNI DNI
ler By: Jump to:	Active
oop Structure V 0x0 Jump	
<pre>japerf:omp_fn.1:B25954 (9EA91: mov %rax, 0xfffffb0(%rbp) </pre>	
rajaperf:omp_fn.1:B25955 (loop_header) loop_1:1/2 0x9EA95: mov \$0x0, 0xfffffb8(%rbp) 0x9EA9D: nop	
<pre>rajaperf:omp_fn.1:B25956 (loop_header)loop_1.1:1/3 0x9EA9E: mov 0xfffffb8(%rbp) 0, %rax 0x9EAA2: cmp 0xfffffd8(%rbp) num_0, %rax 0x9EAA6: jl rajaperf:omp_fn.1:B25959 rajaperf:omp_fn.1:B25957 </pre>	
rajaperf::omp_fn.1:B25959 loop_1.1: 2/3 0x9EAB5: mov \$0x0, 0xfffffc0(%rbp)	Loop Structure



Problem Statement

Program analysts spend huge amount of time navigating large binary codes to improve compiler performance. Assembly code is difficult to understand for lack of intuitive correlation with the source code, hampering compiler optimization.

Solution: *DisViz* aids understanding through interactive visual analysis of source and assembly code relationships.

Novel Layout of Loops in Disassembly



Providing Structure to Disassembly Code

We aim to support large binaries. The example above has more than 260,000 assembly instructions, 1,000 source files & 3,000 loops.



Design phase of pseudo blocks solving corner cases.



Assembly Code with Basic blocks

(c) Our layout

We visualize assembly code structured in basic blocks:

Basic Block Structure



Designing Orders of Disassembly View

(b) Uncommon case

After pseudo blocks, it was observed that another order was necessary to understand control flow. The example above shows the 2 orders: Memory Address: The original order in which the block appears in the memory layout.

(a) Ideal case

Loop Structure: The designed order with loop nesting.



