A Plan to Fix Local Variable Debug Information in GCC

Alexandre Oliva

aoliva@redhat.com

`http://people.redhat.com/~aoliva/`

GCC Summit, June, 2008

Summary

```
> -Wall -WTF -O2 -g --no-gOOd
```

gccsum2k8-1st.d:14:15: warning: missing braces around volatile *dwarfs* with no class, way *out of line*, and known to exhibit *unspecified behavior* in multiple *contexts*. Step into this *scope* at your own peril. Watch out and continue, or break off and be finished: unfriendly *bugs* will constantly fly right over your head.

You have been warned!

— unsigned

```
[you@entry point]$ gdb hell█
```

- Why?
- What?
- How?
- Huh?
- How much?
- Where?
- When?

Why?

- More optimizations ⇒ worse debug info
- Could optimize further, given infrastructure
- Can't, won't rebuild without optimization
- Interactive and **postmortem** debugging
- Monitoring in **production**
- Bad compiler output breaks systems
- Better **miss** than **break**

What?

- Better user experience
- Correctness: no misleading information
- Completeness: gone means gone
  - Multiple locations
  - Computed expressions
- No effect on executable code
- No penalty when disabled
- Compelling trade-off when enabled

How?

- Start early, when IR $\cong$ source
- Keep the mapping accurate
  - Leveraging optimizers
- Don't throw the baby away
  - Remember the source, Luke...
- "Value numbering" in var-tracking
  - Multiple locations
  - Computable values

May I ‘c’ an example?

```
find_val (c, v, e) {
  while (c < e) {

    if (c->v == v)
      return c;
    c++;
  }
  return NULL;
}
```

```
find_prev (c, w) {
  while (c) {
    o = c;
    c = c->n;
    if (c == w)
      return o;
  }
  return NULL;
}
```

Abstract

```
  while (?) {

    S{c}
    c = N(c);
  }
```

```
  while (?) {
    o = c;
    c = N(c);
    S{o c}
  }
```

Notation

N(c) for c’s **N**ext.

S{*vars*} for
Arbitrary
Statement
Sequence that
references *vars*.

May I ‘c’ (2) examples?

```
check_arr (c, t) {
  while (c < t) {

    if (c->v > (c+1)->v)
      return c;
    c++;
  }
  return NULL;
}
```

```
check_list (c, t) {
  while (c != t) {
    n = c->n;
    if (c->v > n->v)
      return c;
    c = n;
  }
  return NULL;
}
```

Abstract

```
  while (?) {

    S{c N(c)}
    c = N(c);
  }
```

```
  while (?) {
    n = N(c);
    S{c n}
    c = n;
  }
```

May I ‘c’ (3) examples?

find_val	check_arr	check_list	find_prev
while (?) {	while (?) {	while (?) {	while (?) {

S{c}
c = N(c);
	S{c N(c)}
	c = N(c);
		o = c;
		n = N(c);
		S{**+**c n **o**}
		c = n;
			o = c;
			c = N(c);
			S{o c}

}	}	}	}

Gimplifying

	goto T;	goto T;	goto T;	goto T;

L:	S{c}
	c = N(c);
		? = N(c);
		S{c **?**}
		c = N(c);
			o = c;
			n = N(c);
			S{+c n o}
			c = n;
				o = c;
				c = N(c);
				S{o c}

T:	if (?) goto L;	if (?) goto L;	if (?) goto L;	if (?) goto L;

May I ‘c’ (4) examples?

	find_val	check_arr	check_list	find_prev
L:			o = c;	o = c;
		? = N(c);	n = N(c);	c = N(c);
	S{c}	S{c ?}	S{+c n o}	S{o c}
	c = N(c);	c = N(c);	c = n;	
T:	if (?) goto L;			

Into SSA

L:			$o_6 = c_1$;	$o_5 = c_1$;
		$?_4 =$ N(c_1);	$n_4 =$ N(c_1);	$c_4 =$ N(c_1);
	S{c_1}	S{c_1 $?_4$}	S{c_1 n_4 o_6}	S{o_5 c_4}
	$c_4 =$ N(c_1);	$c_5 = ?_4$;	$c_5 = n_4$;	
T:	$c_1 = \phi$ (c_2(D), $c_{4,5,5,4}$(L)); if (?) goto L;			

What does the user expect to ‘c’?

L:	$\mathrm{S}\{c_1\}$ $c_4 = \mathrm{N}(c_1);$	$?_4 = \mathrm{N}(c_1);$ $\mathrm{S}\{c_1 \; ?_4\}$ $c_5 = ?_4;$	$o_6 = c_1;$ $n_4 = \mathrm{N}(c_1);$ $\mathrm{S}\{c_1 \; n_4 \; o_6\}$ $c_5 = n_4;$	$o_5 = c_1;$ $c_4 = \mathrm{N}(c_1);$ $\mathrm{S}\{o_5 \; c_4\}$
T:	$c_1 = \phi\,(c_2(\mathrm{D}),\, c_{4,5,5,4}(\mathrm{L}));$			

Optimized, using SSA base names

L:	$\mathrm{S}\{c_1\}$ $c_4 = \mathrm{N}(c_1);$	$c_4 = \mathrm{N}(c_1);$ $\mathrm{S}\{c_1 \; c_4\}$	$c_4 = \mathrm{N}(c_1);$ $\mathrm{S}\{c_1 \; c_4\}$	$c_4 = \mathrm{N}(c_1);$ $\mathrm{S}\{c_1 \; c_4\}$
T:	$c_1 = \phi\,(c_2(\mathrm{D}),\, c_4(\mathrm{L}));$			

- Coalescing (inline), propagating copies
- Same representation for different sources
- No way left to tell the right ‘**c**’ in ‘**S**’, put up?

What are we missing?

L:	$\mathsf{S}\{c_1\}$ $c_4 = \mathsf{N}(c_1);$	$?_4 = \mathsf{N}(c_1);$ $\mathsf{S}\{c_1\ ?_4\}$ $c_5 = ?_4;$	$o_6 = c_1;$ $n_4 = \mathsf{N}(c_1);$ $\mathsf{S}\{c_1\ n_4\ o_6\}$ $c_5 = n_4;$	$o_5 = c_1;$ $c_4 =$ **[n =]** $\mathsf{N}(c_1);$ $\mathsf{S}\{o_5\ c_4\}$
T:	$c_1 = \phi\ (c_2(\mathsf{D}),\ c_{4,5,5,4}(\mathsf{L}));$			

DEF-to-DECL map

L:	$\mathsf{S}\{c_1\}$ $c_4 = \mathsf{N};\ \vert$ **c**	$c_4 = \mathsf{N};\ \vert$ **c??** $\mathsf{S}\{c_1\ c_4\}$	$c_4 = \mathsf{N};\ \vert$ **n c??** $\mathsf{S}\{c_1\ c_4\}$	$c_4 = \mathsf{N};\ \vert$ **[n] c** $\mathsf{S}\{c_1\ c_4\}$
T:	$c_1 = \phi;\ \vert$ **c**	$c_1 = \phi;\ \vert$ **c**	$c_1 = \phi;\ \vert$ **c o??**	$c_1 = \phi;\ \vert$ **c o??**

- Back-propagating deleted assignments
- P.G.Armour's 2OI: can't know you don't know
- Fragile 1,2:{N}, ambiguous (3,4**[n=]**:n≡c)

Aren't we missing the point?

L:	S{c_1} c_4 = N(c_1);	$?_4$ = N(c_1); S{c_1 $?_4$} $c_5 = ?_4$;	$o_6 = c_1$; n_4 = N(c_1); S{c_1 n_4 o_6} $c_5 = n_4$;	$o_5 = c_1$; c_4 = **[n =]** N(c_1); S{o_5 c_4}
T:	$c_1 = \phi\ (c_2(\mathsf{D}), c_{4,5,5,4}(\mathsf{L}))$;			

DEF-to-(DECL, bind point) map

L:	S{c_1} c_4 = N; | **c**	c_4 = N; | **c**P_1 S{c_1 c_4} # P_1	# P_1 c_4 = N; | **n c**P_2 S{c_1 c_4} # P_2	# P_1 c_4 = N; | **[n] c** S{c_1 c_4}
T:	$c_1 = \phi$; | **c**	$c_1 = \phi$; | **c**	$c_1 = \phi$; | **c o**P_1	$c_1 = \phi$; | **c o**P_1

- Replace removed copies with bind points
- Correct, Complete, Complex & Co
- Copying, removing, adjusting bind points

You know what?

L:	$\mathsf{S}\{c_1\}$ $c_4 = \mathsf{N}(c_1);$	$?_4 = \mathsf{N}(c_1);$ $\mathsf{S}\{c_1\ ?_4\}$ $c_5 = ?_4;$	$o_6 = c_1;$ $n_4 = \mathsf{N}(c_1);$ $\mathsf{S}\{c_1\ n_4\ o_6\}$ $c_5 = n_4;$	$o_5 = c_1;$ $c_4 = \mathbf{[n\ =]}\ \mathsf{N}(c_1);$ $\mathsf{S}\{o_5\ c_4\}$
T:	$c_1 = \phi\ (c_2(\mathsf{D}),\ c_{4,5,5,4}(\mathsf{L}));$			

DECL-to-DEF at bind point

L:	$\mathsf{S}\{c_1\}$ $c_4 = \mathsf{N};\ \mid\ \mathbf{c}$	$c_4 = \mathsf{N};$ $\mathsf{S}\{c_1\ c_4\}$ $\#\ \mathbf{c} \Rightarrow c_4$	$\#\ \mathbf{o} \Rightarrow c_1$ $c_4 = \mathsf{N};\ \mid\ \mathbf{n}$ $\mathsf{S}\{c_1\ c_4\}$ $\#\ \mathbf{c} \Rightarrow c_4$	$\#\ \mathbf{o} \Rightarrow c_1$ $c_4 = \mathsf{N};\ \mid\ \mathbf{[n]\ c}$ $\mathsf{S}\{c_1\ c_4\}$
T:	$c_1 = \phi;\ \mid\ \mathbf{c}$	$c_1 = \phi;\ \mid\ \mathbf{c}$	$c_1 = \phi;\ \mid\ \mathbf{c}$	$c_1 = \phi;\ \mid\ \mathbf{c}$

- Bind points are effectively uses!
- Optimizers know how to update them
- Handling arbitrary expressions, losing track

How much?

- No penalty when disabled
- Memory
  - Don't forget too early
  - Should not explode memory use
  - Savings in var-tracking and SSA coalescing
- Performance
  - Must not affect optimizations
  - Should not make compiler too slow

How little?

- Reuse of infrastructure
  - New code mostly in var-tracking
  - Simple localized changes elsewhere
    - Most trivial, without performance impact
- Minimalistic simplicity
- Little maintenance burden
  - Automated regression testing
- Alternate representations for lower footprint?

Where? When?

- Prototype (?) development underway
- var-tracking-assignments-branch (4.3ish)
- Variations, experiments, bugs, features
- Too early for demo, “works” for toy cases
- Infrastructure and further improvements (4.4)
- **Theory** (design) vs. practice (branch)

What else?