LTO Early Debug

Improving debug information when using LTO

Richard Biener
SUSE Labs
rguenther@suse.com

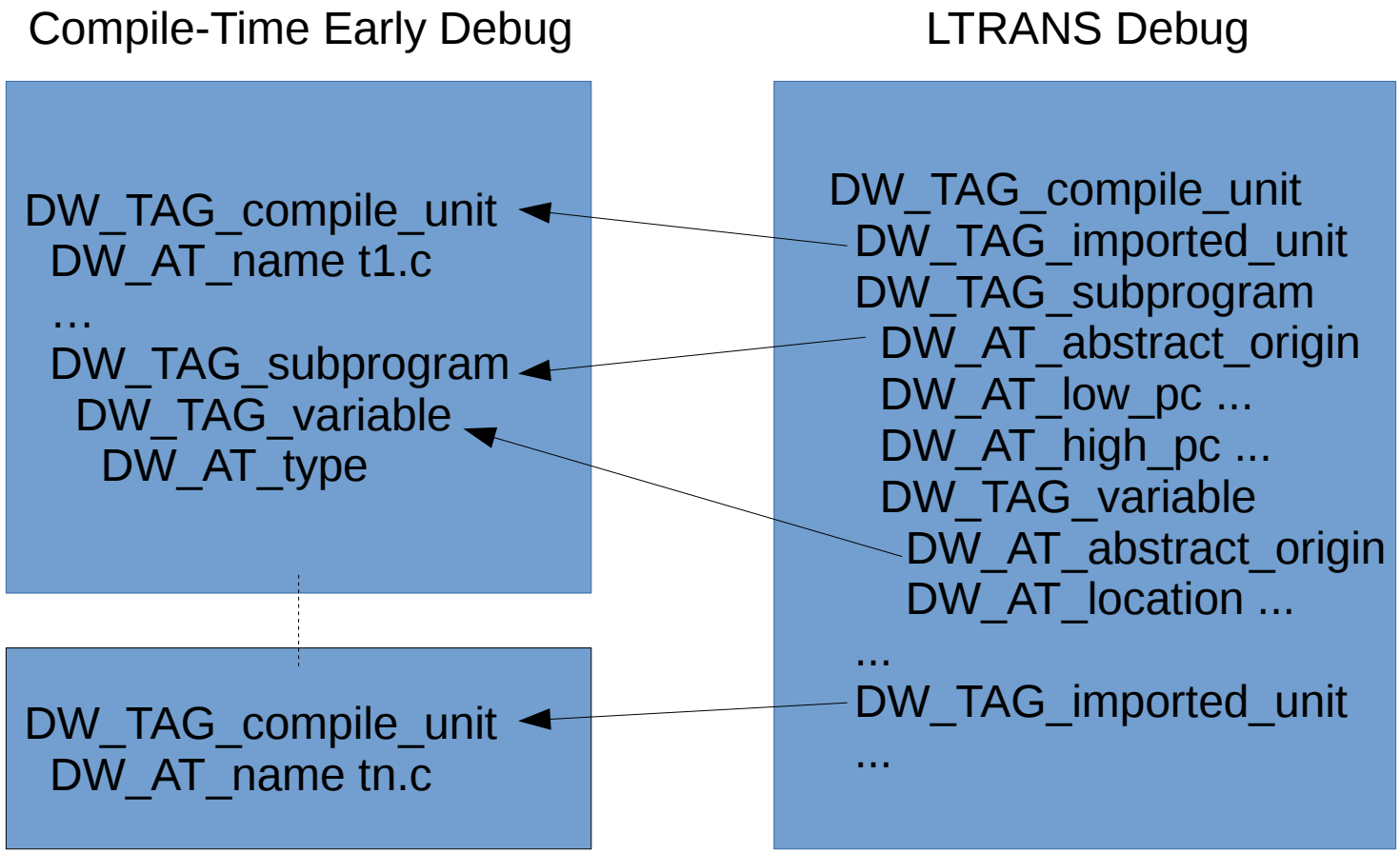
Current Status

The debug experience when using LTO is far from optimal, especially when using non-C programs. Generally the experience is more like debugging such programs as if they were written in C. For example

- libstdc++-v3 pretty printers will not work
- Setting break points on methods is difficult
- Macro information is unrecoverably lost

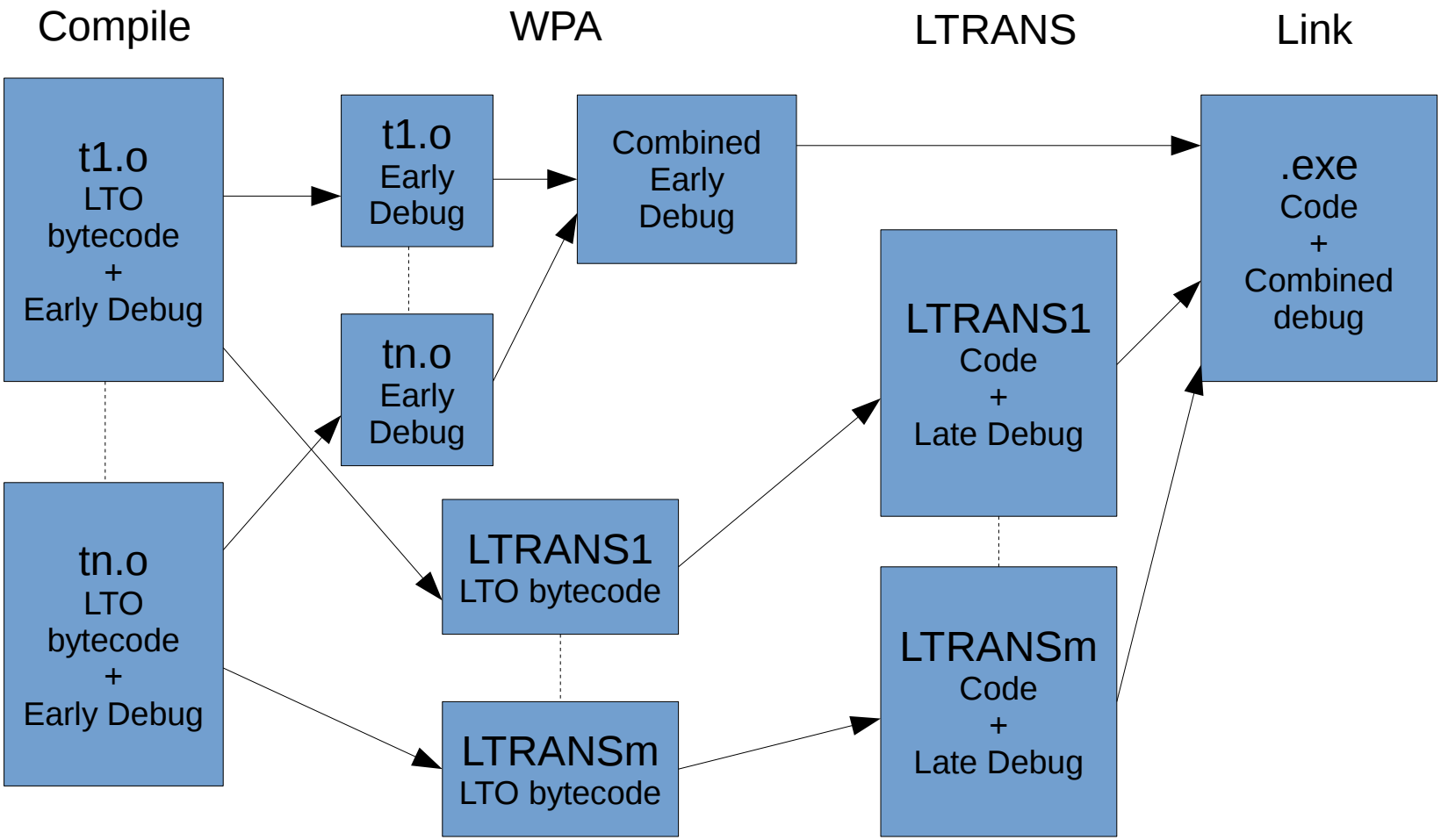
Proposal

The proposal is to leverage the early debug work, splitting the LTO DWARF into two pieces.



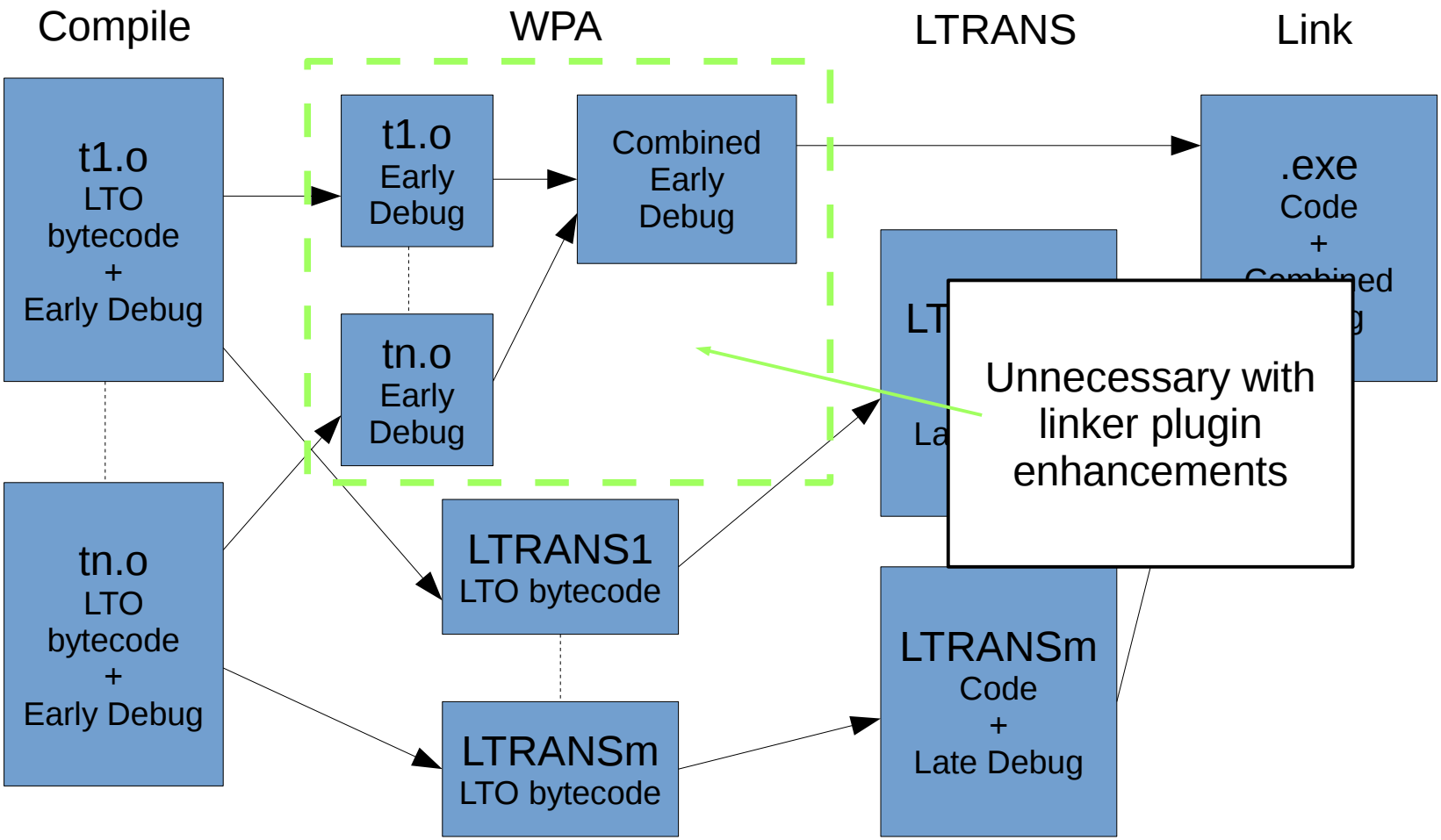
Proposal, cont.

On the tooling side the current setup aimed for backward compatibility is



Proposal, cont.

On the tooling side the current setup aimed for backward compatibility is



Result

```
 Running target unix/-flto/-g/
 FAIL: 23_containers/deque/cons/clear_allocator.cc execution test
-FAIL: libstdc++-prettyprinters/48362.cc print t1
-FAIL: libstdc++-prettyprinters/48362.cc print t2
-FAIL: libstdc++-prettyprinters/cxx11.cc print efl
...
-FAIL: libstdc++-prettyprinters/whatis.cc whatis unord2_holder
-FAIL: libstdc++-prettyprinters/whatis.cc whatis ustring_holder

    === libstdc++ Summary for unix/-flto/-g/ ===

 # of expected passes        11423
-# of unexpected failures 140
+# of unexpected failures    1
 # of expected failures        65
-# of unsupported tests        234
+# of unsupported tests        243
```

Result

	C++ Hello early	late	non-LTO	Tramp3d -Ofast early	late	non-LTO
.debug_info	302e	c6	308b	1ae616	4349f	38f00f
.debug_abbrev	51b	69	542	f8e	4f8	1590
.debug_str	2605		25ff	512c2a	51cb	51309f
.debug_aranges		30	30		50	33c0
.debug_line		73	2ac		2e15	2f52a
.debug_loc					279e3	261cfd

General issues with Early Debug

There are several caveats with generating DIEs early, especially if they are output into the final object in the early form.

Early DIEs may not refer to objects that may be optimized out later. This frequently happens with const objects and their initializers. When the initializer contains only literal constants we can use DW_AT_const_value instead of a location. But symbolic constants cannot be handled this way.

```
static const char a[] = “Hello World”;
static const char *p = a;
```

The immediate solution is to defer outputting initializers to the late phase. This is not always possible, esp. for language specific places we might end up with such references in

```
void f (void);
template <void (*X)(void)> class X;
X <f>;
```

General issues with Early Debug

Entities like VLAs have context dependent types – they are copied by inlining. There is currently no way to add a location to the VLA types DW_AT_upper_bound late without copying the type DIEs and creating references to that copied type.

A DWARF Extension?

A DWARF extension to specify locations symbolically might come handy here.

- DW_OP_die_addr – put the location of another DIE on the stack (kind of a procedure call)
- DW_OP_lookup_die_val – same as above but perform a lookup of the actual DIE in the bindings of the current evaluation and finish the exec like DW_OP_stack_val

The first form can be used for static initializers refering to symbolic addresses. The second form can be used for context dependent entities like VLAs. The VLA DW_AT_upper_bound in the abstract copy can refer to an abstract DIE via DW_OP_lookup_die_val. In the concrete instance we only need the VLA decl DIE refering to the abstract copy and have a concrete instance of the abstract DIE used in DW_AT_upper_bound. The consumer would lookup those (based on refering to the looked up DIE via DW_AT_abstract_origin).

TODO

Things remaining TODO for getting this into GCC 7

- Implement the early LTO debug section copying for non-ELF simple-object formats COFF, XCOFF and MACH-O.
- Get the dwarf2out.c changes reviewed. Split out parts that are independent.
- Get non-C/C++ debugging tried out.

Opportunities

Opportunities opened up by landing this on trunk

- Perform free-lang-data unconditionally (not really dependent on early *LTO* debug)
- Do a lot less LTO streaming of types and associated data that is only required to create debug information.
- Actually **drop** this stuff (in free-lang-data) from GIMPLE.

Q&A


SUSE
We adapt. You succeed.