sparse Documentation

Release 0.6.2

sparse's development community

Aug 05, 2020

Contents

CHAPTER 1

About Sparse

Sparse, the semantic parser, provides a compiler frontend capable of parsing most of ANSI C as well as many GCC extensions, and a collection of sample compiler backends, including a static analyzer also called *sparse*. Sparse provides a set of annotations designed to convey semantic information about types, such as what address space pointers point to, or what locks function acquires or releases.

Linus Torvalds started writing Sparse in 2003, initially targeting issues such as mixing pointers to user address space and pointers to kernel address space.

Josh Triplett was Sparse's first maintainer in 2006. This role was taken over by Christopher Li in 2009 and by Luc Van Oostenryck in late 2018.

CHAPTER 2

Getting Sparse

You can find tarballs of released versions of Sparse at https://www.kernel.org/pub/software/devel/sparse/dist/.

The most recent version can be obtained directly from the Git repository with the command:

```
git clone git://git.kernel.org/pub/scm/devel/sparse/sparse.git
```

You can also browse the Git repository or use the mirror at https://github.com/lucvoo/sparse.

Once you have the sources, to build Sparse and install it in your ~/bin directory, just do:

```
cd sparse
make
make install
```

To install it in another directory, use:

```
make PREFIX=<some directory> install
```

CHAPTER 3

Contributing and reporting bugs

Submission of patches and reporting of bugs, as well as discussions related to Sparse, should be done via the mailing list: linux-sparse@vger.kernel.org. You do not have to be subscribed to the list to send a message there. Previous discussions and bug reports are available on the list archives at https://marc.info/?l=linux-sparse.

To subscribe to the list, send an email with `subscribe linux-sparse` in the body to `majordomo@vger.kernel.org`.

Bugs can also be reported and tracked via the Linux kernel's bugzilla for sparse.

CHAPTER 4

User documentation

4.1 __nocast vs __bitwise

`__nocast` warns about explicit or implicit casting to different types. HOWEVER, it doesn't consider two 32-bit integers to be different types, so a `__nocast int` type may be returned as a regular `int` type and then the `__nocast` is lost.

So `__nocast` on integer types is usually not that powerful. It just gets lost too easily. It's more useful for things like pointers. It also doesn't warn about the mixing: you can add integers to `__nocast` integer types, and it's not really considered anything wrong.

`__bitwise` ends up being a *stronger integer separation*. That one doesn't allow you to mix with non-bitwise integers, so now it's much harder to lose the type by mistake.

So the basic rule is:

- `__nocast` on its own tends to be more useful for *big* integers that still need to act like integers, but you want to make it much less likely that they get truncated by mistake. So a 64-bit integer that you don't want to mistakenly/silently be returned as `int`, for example. But they mix well with random integer types, so you can add to them etc without using anything special. However, that mixing also means that the `__nocast` really gets lost fairly easily.
- `__bitwise` is for *unique types* that cannot be mixed with other types, and that you'd never want to just use as a random integer (the integer `0` is special, though, and gets silently accepted - it's kind of like `NULL` for pointers). So `gfp_t` or the `safe endianness` types would be `__bitwise`: you can only operate on them by doing specific operations that know about *that* particular type.

Generally, you want `__bitwise` if you are looking for type safety. `__nocast` really is pretty weak.

4.1.1 Reference:

- Linus' e-mail about `__nocast` vs `__bitwise`:

  https://marc.info/?l=linux-mm&m=133245421127324&w=2

CHAPTER 5

Developer documentation

5.1 Test suite

Sparse has a number of test cases in its validation directory. The test-suite script aims at making automated checking of these tests possible. It works by embedding tags in C comments in the test cases.

5.1.1 Tag's syntax

`check-name:` *name*

Name of the test. This is the only mandatory tag.

`check-description:` *description ...*

A description of what the test checks.

`check-command:` *command arg ...*

There are different kinds of tests. Some can validate the sparse preprocessor, while others will use sparse, cgcc, or even other backends of the library. check-command allows you to give a custom command to run the test-case. The `$file` string is special. It will be expanded to the file name at run time. It defaults to `sparse $file`.

`check-arch-ignore:` *arch[|...]*

`check-arch-only:` *arch[|...]*

Ignore the test if the current architecture (as returned by `uname -m`) matches or not one of the archs given in the pattern.

`check-assert:` *condition*

Ignore the test if the given condition is false when evaluated as a static assertion (`_Static_assert`).

`check-cpp-if:` *condition*

Ignore the test if the given condition is false when evaluated by sparse's pre-processor.

`check-exit-value:` *value*

The expected exit value of check-command. It defaults to 0.

`check-timeout: timeout`

The maximum expected duration of check-command, in seconds. It defaults to 1.

`check-output-start` / `check-output-end`

The expected output (stdout and stderr) of check-command lies between those two tags. It defaults to no output.

`check-output-ignore` / `check-error-ignore`

Don't check the expected output (stdout or stderr) of check-command (useful when this output is not comparable or if you're only interested in the exit value). By default this check is done.

`check-known-to-fail`

Mark the test as being known to fail.

`check-output-contains: pattern`

Check that the output (stdout) contains the given pattern. Several such tags can be given, in which case the output must contains all the patterns.

`check-output-excludes: pattern`

Similar than the above one, but with opposite logic. Check that the output (stdout) doesn't contain the given pattern. Several such tags can be given, in which case the output must contains none of the patterns.

`check-output-pattern(nbr): pattern`

`check-output-pattern(min,max): pattern`

Similar to the contains/excludes above, but with full control of the number of times the pattern should occur in the output. If *min* or *max* is − the corresponding check is ignored.

5.1.2 Using test-suite

The test-suite script is called through the check target of the Makefile. It will try to check every test case it finds (`find validation -name '*.c'`). It can be called to check a single test with:

```
$ cd validation
$ ./test-suite single preprocessor/preprocessor1.c
     TEST     Preprocessor #1 (preprocessor/preprocessor1.c)
preprocessor/preprocessor1.c passed !
```

5.1.3 Writing a test

The test-suite comes with a format command to make a test easier to write:

```
test-suite format [-a] [-l] [-f] file [name [cmd]]
```

name: **check-name value** If no name is provided, it defaults to the file name.

cmd: **check-command value** If no cmd is provided, it defaults to `sparse $file`.

The output of the test-suite format command can be redirected into the test case to create a test-suite formatted file.:

```
$ ./test-suite format bad-assignment.c Assignment >> bad-assignment.c
$ cat !$
cat bad-assignment.c
/*
 * check-name: bad assignment
 *
```

(continues on next page)

(continued from previous page)

```
 * check-command: sparse $file
 * check-exit-value: 1
 *
 * check-output-start
bad-assignment.c:3:6: error: Expected ; at end of statement
bad-assignment.c:3:6: error: got \
 * check-output-end
 */
```

The same effect without the redirection can be achieved by using the `-a` option.

You can define the check-command you want to use for the test.:

```
$ ./test-suite format -a validation/preprocessor2.c "Preprocessor #2" \
                "sparse -E \$file"
$ cat !$
cat validation/preprocessor2.c
/*
 * This one we happen to get right.
 *
 * It should result in a simple
 *
 *      a + b
 *
 * for a proper preprocessor.
 */
#define TWO a, b

#define UNARY(x) BINARY(x)
#define BINARY(x, y) x + y

UNARY(TWO)
/*
 * check-name: Preprocessor #2
 *
 * check-command: sparse -E $file
 * check-exit-value: 0
 *
 * check-output-start

a + b
 * check-output-end
 */
```

5.2 sparse - extra options for developers

5.2.1 SYNOPSIS

`tools` [*options*]... *file.c*‘

5.2.2 DESCRIPTION

This file is a complement of sparse’s man page meant to document options only useful for development on sparse itself.

5.2.3 OPTIONS

-fdump-ir=pass,[pass]
Dump the IR at each of the given passes.

The passes currently understood are:

- `linearize`
- `mem2reg`
- `final`

The default pass is `linearize`.

-f<name-of-the-pass>[-disable|-enable|=last]
If `=last` is used, all passes after the specified one are disabled. By default all passes are enabled.

The passes currently understood are:

- `linearize` (can't be disabled)
- `mem2reg`
- `optim`

-vcompound
Print all compound global data symbols with their sizes and alignment.

-vdead
Add `OP_DEATHNOTE` annotations to dead pseudos.

-vdomtree
Dump the dominance tree after its calculation.

-ventry
Dump the IR after all optimization passes.

-vpostorder
Dump the reverse postorder traversal of the CFG.

5.3 Sparse API

5.3.1 Utilities

Pointer list manipulation

int **ptr_list_size** (struct ptr_list **head*)
Get the size of a ptrlist.

Parameters

- **head** – the head of the list

Returns the size of the list given by **head**.

bool **ptr_list_empty** (const struct ptr_list **head*)
Test if a list is empty.

Parameters

- **head** – the head of the list

Returns `true` if the list is empty, `false` otherwise.

bool **ptr_list_multiple** (const struct ptr_list **head*)
Test is a list contains more than one element.

Parameters

- **head** – the head of the list

Returns `true` if the list has more than 1 element, `false` otherwise.

void ***first_ptr_list** (struct ptr_list **head*)
Get the first element of a ptrlist.

Parameters

- **head** – the head of the list

Returns the first element of the list or `NULL` if the list is empty

void ***last_ptr_list** (struct ptr_list **head*)
Get the last element of a ptrlist.

Parameters

- **head** – the head of the list

Returns the last element of the list or `NULL` if the list is empty

void ***ptr_list_nth_entry** (struct ptr_list **list*, unsigned int *idx*)
Get the nth element of a ptrlist.

Parameters

- **head** – the head of the list

Returns the nth element of the list or `NULL` if the list is too short.

int **linearize_ptr_list** (struct ptr_list **head*, void ***arr*, int *max*)
Linearize the entries of a list.

Parameters

- **head** – the list to be linearized
- **arr** – a `void*` array to fill with **head**'s entries
- **max** – the maximum number of entries to store into **arr**

Returns the number of entries linearized.

Linearize the entries of a list up to a total of **max**, and return the nr of entries linearized.

The array to linearize into (**arr**) should really be `void *x[]`, but we want to let people fill in any kind of pointer array, so let's just call it `void **`.

void **pack_ptr_list** (struct ptr_list **listp*)
Pack a ptrlist.

Parameters

- **listp** – a pointer to the list to be packed.

When we've walked the list and deleted entries, we may need to re-pack it so that we don't have any empty blocks left (empty blocks upset the walking code).

void **split_ptr_list_head** (struct ptr_list *head)
Split a ptrlist block.

Parameters

- **head** – the ptrlist block to be split

A new block is inserted just after **head** and the entries at the half end of **head** are moved to this new block. The goal being to create space inside **head** for a new entry.

void ****__add_ptr_list** (struct ptr_list ***listp*, void **ptr*)
Add an entry to a ptrlist.

Parameters

- **listp** – a pointer to the list
- **ptr** – the entry to add to the list

Returns the address where the new entry is stored.

Note code must not use this function and should use `add_ptr_list()` instead.

void ****__add_ptr_list_tag** (struct ptr_list ***listp*, void **ptr*, unsigned long *tag*)
Add a tagged entry to a ptrlist.

Parameters

- **listp** – a pointer to the list
- **ptr** – the entry to add to the list
- **tag** – the tag to add to **ptr**

Returns the address where the new entry is stored.

Note code must not use this function and should use `add_ptr_list_tag()` instead.

bool **lookup_ptr_list_entry** (const struct ptr_list **head*, const void **entry*)
Test if some entry is already present in a ptrlist.

Parameters

- **list** – the head of the list
- **entry** – the entry to test

Returns `true` if the entry is already present, `false` otherwise.

int **delete_ptr_list_entry** (struct ptr_list ***list*, void **entry*, int *count*)
Delete an entry from a ptrlist.

Parameters

- **list** – a pointer to the list
- **entry** – the item to be deleted
- **count** – the minimum number of times **entry** should be deleted or 0.

int **replace_ptr_list_entry** (struct ptr_list ***list*, void **old_ptr*, void **new_ptr*, int *count*)
Replace an entry in a ptrlist.

Parameters

- **list** – a pointer to the list
- **old_ptr** – the entry to be replaced
- **new_ptr** – the new entry
- **count** – the minimum number of times **entry** should be deleted or 0.

void * **undo_ptr_list_last** (struct ptr_list **head*)
Remove the last entry of a ptrlist.

Parameters

- **head** – a pointer to the list

Returns the last element of the list or NULL if the list is empty.

Note this doesn't repack the list

void * **delete_ptr_list_last** (struct ptr_list **head*)
Remove the last entry and repack the list.

Parameters

- **head** – a pointer to the list

Returns the last element of the list or NULL if the list is empty.

void **concat_ptr_list** (struct ptr_list **a*, struct ptr_list ***b*)
Concat two ptrlists.

Parameters

- **a** – the source list
- **b** – a pointer to the destination list.

The element of **a** are added at the end of **b**.

void **copy_ptr_list** (struct ptr_list ***listp*, struct ptr_list **src*)
Copy the elements of a list at the end of another list.

Parameters

- **listp** – a pointer to the destination list.
- **src** – the head of the source list.

void **__free_ptr_list** (struct ptr_list ***listp*)
Free a ptrlist.

Parameters

- **listp** – a pointer to the list

Each blocks of the list are freed (but the entries themselves are not freed).

Note code must not use this function and should use the macro `free_ptr_list()` instead.

Miscellaneous utilities

unsigned int **hexval** (unsigned int *c*)
Return the value coresponding to an hexadecimal digit.

void ***xmemdup** (const void **src*, size_t *len*)
Duplicate a memory buffer in a newly allocated buffer.

Parameters

- **src** – a pointer to the memory buffer to be duplicated
- **len** – the size of the memory buffer to be duplicated

Returns a pointer to a copy of **src** allocated via `__alloc_bytes()`.

char *__xstrdup__(const char *_src_)
Duplicate a null-terminated string in a newly allocated buffer.

Parameters

- **src** – a pointer to string to be duplicated

Returns a pointer to a copy of **str** allocated via `__alloc_bytes()`.

char *__xasprintf__(const char *_fmt_, ...)
Printf to allocated string.

Parameters

- **fmt** – the format followed by its arguments.

Returns the allocated & formatted string.

This function is similar to asprintf() but the resulting string is allocated with __alloc_bytes().

char *__xvasprintf__(const char *_fmt_, va_list _ap_)
Vprintf to allocated string.

Parameters

- **fmt** – the format
- **ap** – the variadic arguments

Returns the allocated & formatted string.

This function is similar to asprintf() but the resulting string is allocated with __alloc_bytes().

5.3.2 Parsing

Constant expression values

int __is_zero_constant__(struct expression *_expr_)
Test if an expression evaluates to the constant `0`.

Returns `1` if **expr** evaluate to `0`, `0` otherwise.

int __expr_truth_value__(struct expression *_expr_)
Test the compile time truth value of an expression.

Returns

- `-1` if **expr** is not constant,
- `0` or `1` depending on the truth value of **expr**.

5.3.3 Typing

struct symbol *__evaluate_expression__(struct expression *_expr_)
Evaluate the type of an expression.

Parameters

- **expr** – the expression to be evaluated

Returns the type of the expression or `NULL` if the expression can't be evaluated

struct symbol *__evaluate_statement__(struct statement *_stmt_)
Evaluate the type of a statement.

Parameters

- **stmt** – the statement to be evaluated

Returns the type of the statement or NULL if it can't be evaluated

void **evaluate_symbol_list** (struct symbol_list **list*)
Evaluate the type of a set of symbols.

Parameters

- **list** – the list of the symbol to be evaluated

5.3.4 Optimization

Instruction simplification

Notation

The following conventions are used to describe the simplications:

- Uppercase letters are reserved for constants:
  - *M* for a constant mask,
  - *S* for a constant shift,
  - *N* for a constant number of bits (usually other than a shift),
  - *C* or 'K' for others constants.
- Lowercase letters *a*, *b*, *x*, *y*, ... are used for non-constants or when it doesn't matter if the pseudo is a constant or not.
- Primes are used if needed to distinguish symbols (*M*, *M'*, ...).
- Expressions or sub-expressions involving only constants are understood to be evaluated.
- *$mask(N)* is used for *((1 << N) -1)*
- *$trunc(x, N)* is used for *(x & $mask(N))*
- Expressions like *(-1 << S)*, *(-1 >> S)* and others formulae are understood to be truncated to the size of the current instruction (needed, since in general this size is not the same as the one used by sparse for the evaluation of arithmetic operations).
- *TRUNC(x, N)* is used for a truncation *to* a size of *N* bits
- *ZEXT(x, N)* is used for a zero-extension *from* a size of *N* bits
- *OP(x, C)* is used to represent some generic operation using a constant, including when the constant is implicit (e.g. *TRUNC(x, N)*).
- *MASK(x, M)* is used to respresent a 'masking' instruction:
  - *AND(x, M)*
  - *LSR(x, S)*, with *M* = (-1 << S)
  - *SHL(x, S)*, with *M* = (-1 >> S)
  - *TRUNC(x, N)*, with *M* = $mask(N)
  - *ZEXT(x, N)*, with *M* = $mask(N)
- *SHIFT(x, S)* is used for *LSR(x, S)* or *SHL(x, S)*.

Utilities

static struct basic_block *__phi_parent__(struct basic_block *_source_, pseudo_t _pseudo_)
Find the trivial parent for a phi-source.

static int __get_phisources__(struct instruction *_sources[]_, int _nbr_, struct instruction *_insn_)
Copy the phi-node's phisrcs into to given array.

Returns 0 if the the list contained the expected number of element, a positive number if there was more than expected and a negative one if less.

Note we can't reuse a function like linearize_ptr_list() because any VOIDs in the phi-list must be ignored here as in this context they mean 'entry has been removed'.

static pseudo_t __trivial_phi__(pseudo_t _pseudo_, struct instruction *_insn_, struct pseudo_list **_list_)
Detect trivial phi-nodes.

Parameters

- **insn** – the phi-node
- **pseudo** – the candidate resulting pseudo (NULL when starting)

Returns the unique result if the phi-node is trivial, NULL otherwise

A phi-node is trivial if it has a single possible result:

- all operands are the same
- **the operands are themselves defined by a chain or cycle of phi-nodes** and the set of all operands involved contains a single value not defined by these phi-nodes

Since the result is unique, these phi-nodes can be removed.

int __kill_insn__(struct instruction *_insn_, int _force_)
Kill an instruction.

Parameters

- **insn** – the instruction to be killed
- **force** – if unset, the normal case, the instruction is not killed if not free of possible side-effect; if set the instruction is unconditionally killed.

The killed instruction is removed from its BB and the usage of all its operands are removed. The instruction is also marked as killed by setting its ->bb to NULL.

static int __dead_insn__(struct instruction *_insn_, pseudo_t *_src1_, pseudo_t *_src2_, pseudo_t *_src3_)
Kill trivially dead instructions.

static inline int __replace_pseudo__(struct instruction *_insn_, pseudo_t *_pp_, pseudo_t _new_)
Replace the operand of an instruction.

Parameters

- **insn** – the instruction
- **pp** – the address of the instruction's operand
- **new** – the new value for the operand

Returns REPEAT_CSE.

static unsigned int __operand_size__(struct instruction *_insn_, pseudo_t _pseudo_)
Try to determine the maximum size of bits in a pseudo.

Right now this only follow casts and constant values, but we could look at things like AND instructions, etc.

Simplifications

static int **simplify_mask_or_and**(struct instruction **insn*, unsigned long long *mask*, pseudo_t *ora*, pseudo_t *orb*)
Try to simplify MASK(OR(AND(x, M'), b), M).

Parameters

- **insn** – the masking instruction
- **mask** – the associated mask (M)
- **ora** – one of the OR's operands, guaranteed to be PSEUDO_REG
- **orb** – the other OR's operand

Returns 0 if no changes have been made, one or more REPEAT_* flags otherwise.

static int **simplify_mask_or**(struct instruction **insn*, unsigned long long *mask*, struct instruction **or*)
Try to simplify MASK(OR(a, b), M).

Parameters

- **insn** – the masking instruction
- **mask** – the associated mask (M)
- **or** – the OR instruction

Returns 0 if no changes have been made, one or more REPEAT_* flags otherwise.

static int **simplify_mask_shift_or**(struct instruction **sh*, struct instruction **or*, unsigned long long *mask*)
Try to simplify MASK(SHIFT(OR(a, b), S), M).

Parameters

- **sh** – the shift instruction
- **or** – the OR instruction
- **mask** – the mask associated to MASK (M):

Returns 0 if no changes have been made, one or more REPEAT_* flags otherwise.

static int **simplify_memop**(struct instruction **insn*)
Simplify memops instructions.

Note We walk the whole chain of adds/subs backwards. That's not only more efficient, but it allows us to find loops.

static int **simplify_cond_branch**(struct instruction **br*, struct instruction **def*, pseudo_t *newcond*)
Simplify SET_NE/EQ $0 + BR.

5.4 Sparse's Intermediate Representation

5.4.1 Instructions

This document briefly describes which field of struct instruction is used by which operation.

Some of those fields are used by almost all instructions, some others are specific to only one or a few instructions. The common ones are:

- .src1, .src2, .src3: (pseudo_t) operands of binops or ternary ops.
- .src: (pseudo_t) operand of unary ops (alias for .src1).
- .target: (pseudo_t) result of unary, binary & ternary ops, is sometimes used otherwise by some others instructions.

- .cond: (pseudo_t) input operands for condition (alias .src/.src1)
- .type: (symbol*) usually the type of .result, sometimes of the operands

Terminators

OP_RET

Return from subroutine.

- .src : returned value (NULL if void)
- .type: type of .src

OP_BR

Unconditional branch

- .bb_true: destination basic block

OP_CBR

Conditional branch

- .cond: condition
- .type: type of .cond, must be an integral type
- .bb_true, .bb_false: destination basic blocks

OP_SWITCH

Switch / multi-branch

- .cond: condition
- .type: type of .cond, must be an integral type
- .multijmp_list: pairs of case-value - destination basic block

OP_UNREACH

Mark code as unreachable

OP_COMPUTEDGOTO

Computed goto / branch to register

- .src: address to branch to (void*)
- .multijmp_list: list of possible destination basic blocks

Arithmetic binops

They all follow the same signature:

- .src1, .src2: operands (types must be compatible with .target)
- .target: result of the operation (must be an integral type)
- .type: type of .target

OP_ADD

Integer addition.

OP_SUB

Integer subtraction.

OP_MUL

Integer multiplication.

OP_DIVU

Integer unsigned division.

OP_DIVS

Integer signed division.

OP_MODU

Integer unsigned remainder.

OP_MODS

Integer signed remainder.

OP_SHL

Shift left (integer only)

OP_LSR

Logical Shift right (integer only)

OP_ASR

Arithmetic Shift right (integer only)

Floating-point binops

They all follow the same signature:

- .src1, .src2: operands (types must be compatible with .target)
- .target: result of the operation (must be a floating-point type)
- .type: type of .target

OP_FADD

Floating-point addition.

OP_FSUB

Floating-point subtraction.

OP_FMUL

Floating-point multiplication.

OP_FDIV

Floating-point division.

Logical ops

They all follow the same signature:

- .src1, .src2: operands (types must be compatible with .target)
- .target: result of the operation
- .type: type of .target, must be an integral type

OP_AND

Logical AND

OP_OR

Logical OR

OP_XOR

Logical XOR

Integer compares

They all have the following signature:

- .src1, .src2: operands (types must be compatible)
- .target: result of the operation (0/1 valued integer)
- .type: type of .target, must be an integral type

OP_SET_EQ

Compare equal.

OP_SET_NE

Compare not-equal.

OP_SET_LE

Compare less-than-or-equal (signed).

OP_SET_GE

Compare greater-than-or-equal (signed).

OP_SET_LT

Compare less-than (signed).

OP_SET_GT

Compare greater-than (signed).

OP_SET_B

Compare less-than (unsigned).

OP_SET_A

Compare greater-than (unsigned).

OP_SET_BE

Compare less-than-or-equal (unsigned).

OP_SET_AE

Compare greater-than-or-equal (unsigned).

Floating-point compares

They all have the same signature as the integer compares.

The usual 6 operations exist in two versions: 'ordered' and 'unordered'. These operations first check if any operand is a NaN and if it is the case the ordered compares return false and then unordered return true, otherwise the result of the comparison, now guaranteed to be done on non-NaNs, is returned.

OP_FCMP_OEQ

Floating-point compare ordered equal

OP_FCMP_ONE

Floating-point compare ordered not-equal

OP_FCMP_OLE

Floating-point compare ordered less-than-or-equal

OP_FCMP_OGE

Floating-point compare ordered greater-or-equal

OP_FCMP_OLT

Floating-point compare ordered less-than

OP_FCMP_OGT

Floating-point compare ordered greater-than

OP_FCMP_UEQ

Floating-point compare unordered equal

OP_FCMP_UNE

Floating-point compare unordered not-equal

OP_FCMP_ULE

Floating-point compare unordered less-than-or-equal

OP_FCMP_UGE

Floating-point compare unordered greater-or-equal

OP_FCMP_ULT

Floating-point compare unordered less-than

OP_FCMP_UGT

Floating-point compare unordered greater-than

OP_FCMP_ORD

Floating-point compare ordered: return true if both operands are ordered (none of the operands are a NaN) and false otherwise.

OP_FCMP_UNO

Floating-point compare unordered: return false if no operands is ordered and true otherwise.

Unary ops

OP_NOT

Logical not.

- .src: operand (type must be compatible with .target)
- .target: result of the operation
- .type: type of .target, must be an integral type

OP_NEG

Integer negation.

- .src: operand (type must be compatible with .target)
- .target: result of the operation (must be an integral type)
- .type: type of .target

OP_FNEG

Floating-point negation.

- .src: operand (type must be compatible with .target)
- .target: result of the operation (must be a floating-point type)

- .type: type of .target

OP_SYMADDR

Create a pseudo corresponding to the address of a symbol.

- .src: input symbol (must be a PSEUDO_SYM)
- .target: symbol's address

OP_COPY

Copy (only needed after out-of-SSA).

- .src: operand (type must be compatible with .target)
- .target: result of the operation
- .type: type of .target

Type conversions

They all have the following signature:

- .src: source value
- .orig_type: type of .src
- .target: result value
- .type: type of .target

Currently, a cast to a void pointer is treated like a cast to an unsigned integer of the same size.

OP_TRUNC

Cast from integer to an integer of a smaller size.

OP_SEXT

Cast from integer to an integer of a bigger size with sign extension.

OP_ZEXT

Cast from integer to an integer of a bigger size with zero extension.

OP_UTPTR

Cast from pointer-sized unsigned integer to pointer type.

OP_PTRTU

Cast from pointer type to pointer-sized unsigned integer.

OP_PTRCAST

Cast between pointers.

OP_FCVTU

Conversion from float type to unsigned integer.

OP_FCVTS

Conversion from float type to signed integer.

OP_UCVTF

Conversion from unsigned integer to float type.

OP_SCVTF

Conversion from signed integer to float type.

OP_FCVTF

Conversion between float types.

Ternary ops

OP_SEL

- .src1: condition, must be of integral type
- .src2, .src3: operands (types must be compatible with .target)
- .target: result of the operation
- .type: type of .target

OP_RANGE

Range/bounds checking (only used for an unused sparse extension).

- .src1: value to be checked
- .src2, src3: bound of the value (must be constants?)
- .type: type of .src[123]?

Memory ops

OP_LOAD

Load.

- .src: base address to load from
- .offset: address offset
- .target: loaded value
- .type: type of .target

OP_STORE

Store.

- .src: base address to store to
- .offset: address offset
- .target: value to be stored
- .type: type of .target

Others

OP_SETFVAL

Create a pseudo corresponding to a floating-point literal.

- .fvalue: the literal's value (long double)
- .target: the corresponding pseudo
- .type: type of the literal & .target

OP_SETVAL

Create a pseudo corresponding to a string literal or a label-as-value. The value is given as an expression EXPR_STRING or EXPR_LABEL.

- .val: (expression) input expression
- .target: the resulting value

- .type: type of .target, the value

OP_PHI

Phi-node (for SSA form).

- .phi_list: phi-operands (type must be compatible with .target)
- .target: "result"
- .type: type of .target

OP_PHISOURCE

Phi-node source. Like OP_COPY but exclusively used to give a defining instructions (and thus also a type) to *all* OP_PHI operands.

- .phi_src: operand (type must be compatible with .target, alias .src)
- .target: the "result" PSEUDO_PHI
- .type: type of .target
- .phi_users: list of phi instructions using the target pseudo

OP_CALL

Function call.

- .func: (pseudo_t) the function (can be a symbol or a "register", alias .src))
- .arguments: (pseudo_list) list of the associated arguments
- .target: function return value (if any)
- .type: type of .target
- .fntypes: (symbol_list) list of the function's types: the first entry is the full function type, the next ones are the type of each arguments

OP_INLINED_CALL

Only used as an annotation to show that the instructions just above correspond to a function that have been inlined.

- .func: (pseudo_t) the function (must be a symbol, alias .src))
- .arguments: list of pseudos that where the function's arguments
- .target: function return value (if any)
- .type: type of .target

OP_SLICE

Extract a "slice" from an aggregate.

- .base: (pseudo_t) aggregate (alias .src)
- .from, .len: offet & size of the "slice" within the aggregate
- .target: result
- .type: type of .target

OP_ASM

Inlined assembly code.

- .string: asm template
- .asm_rules: asm constraints, rules

Sparse tagging (line numbers, context, whatever)

OP_CONTEXT

Currently only used for lock/unlock tracking.

- .context_expr: unused
- .increment: (1 for locking, -1 for unlocking)
- .check: (ignore the instruction if 0)

Misc ops

OP_ENTRY

Function entry point (no associated semantic).

OP_BADOP

Invalid operation (should never be generated).

OP_NOP

No-op (should never be generated).

OP_DEATHNOTE

Annotation telling the pseudo will be death after the next instruction (other than some other annotation, that is).

5.5 Sparse's Type System

struct symbol is used to represent symbols & types but most parts pertaining to the types are in the field 'ctype'. For the purpose of this document, things can be simplified into:

```
struct symbol {
        enum type type; // SYM_...
        struct ctype {
                struct symbol *base_type;
                unsigned long modifiers;
                unsigned long alignment;
                struct context_list *contexts;
                struct indent *as;
        };
};
```

Some bits, also related to the type, are in struct symbol itself:

- type
- size_bits
- rank
- variadic
- string
- designated_init
- forced_arg
- accessed
- transparent_union

- `base_type` is used for the associated base type.

- `modifiers` is a bit mask for type specifiers (MOD_UNSIGNED, ...), type qualifiers (MOD_CONST, MOD_VOLATILE), storage classes (MOD_STATIC, MOD_EXTERN, ...), as well for various attributes. It's also used internally to keep track of some states (MOD_ACCESS or MOD_ADDRESSABLE).
- `alignment` is used for the alignment, in bytes.
- `contexts` is used to store the informations associated with the attribute `context()`.
- `as` is used to hold the identifier of the attribute `address_space()`.

5.5.1 Kind of types

SYM_BASETYPE

Used by integer, floating-point, void, 'type', 'incomplete' & bad types.

For integer types:

- .ctype.base_type points to `int_ctype`, the generic/abstract integer type
- .ctype.modifiers has MOD_UNSIGNED/SIGNED/EXPLICITLY_SIGNED set accordingly.

For floating-point types:

- .ctype.base_type points to `fp_ctype`, the generic/abstract float type
- .ctype.modifiers is zero.

For the other base types:

- .ctype.base_type is NULL
- .ctype.modifiers is zero.

SYM_NODE

It's used to make variants of existing types. For example, it's used as a top node for all declarations which can then have their own modifiers, address_space, contexts or alignment as well as the declaration's identifier.

Usage:

- .ctype.base_type points to the unmodified type (which must not be a SYM_NODE itself)
- .ctype.modifiers, .as, .alignment, .contexts will contains the 'variation' (MOD_CONST, the attributes, ...).

SYM_PTR

For pointers:

- .ctype.base_type points to the pointee type
- .ctype.modifiers & .as are about the pointee too!

SYM_FN

For functions:

- .ctype.base_type points to the return type
- .ctype.modifiers & .as should be about the function itself but some return type's modifiers creep here (for example, in int foo(void), MOD_SIGNED will be set for the function).

SYM_ARRAY

For arrays:

- .ctype.base_type points to the underlying type
- .ctype.modifiers & .as are a copy of the parent type (and unused)?
- for literal strings, the modifier also contains MOD_STATIC
- sym->array_size is *expression* for the array size.

SYM_STRUCT

For structs:

- .ctype.base_type is NULL
- .ctype.modifiers & .as are not used?
- .ident is the name tag.

SYM_UNION

Same as for structs.

SYM_ENUM

For enums:

- .ctype.base_type points to the underlying type (integer)
- .ctype.modifiers contains the enum signedness
- .ident is the name tag.

SYM_BITFIELD

For bitfields:

- .ctype.base_type points to the underlying type (integer)
- .ctype.modifiers & .as are a copy of the parent type (and unused)?
- .bit_size is the size of the bitfield.

SYM_RESTRICT

Used for bitwise types (aka 'restricted' types):

- .ctype.base_type points to the underlying type (integer)
- .ctype.modifiers & .as are like for SYM_NODE and the modifiers are inherited from the base type with MOD_SPECIFIER removed
- .ident is the typedef name (if any).

SYM_FOULED

Used for bitwise types when the negation op (~) is used and the bit_size is smaller than an `int`. There is a 1-to-1 mapping between a fouled type and its parent bitwise type.

Usage:

- .ctype.base_type points to the parent type
- .ctype.modifiers & .as are the same as for the parent type
- .bit_size is bits_in_int.

SYM_TYPEOF

Should not be present after evaluation:

- .initializer points to the expression representing the type
- .ctype is not used.

Typeofs with a type as argument are directly evaluated during parsing.

SYM_LABEL

Used for labels only.

SYM_KEYWORD

Used for parsing only.

SYM_BAD

Should not be used.

SYM_UNINTIALIZED

Should not be used.

CHAPTER 6

How to contribute

6.1 Submitting patches: the sparse version

Sparse uses a patch submit process similar to the Linux Kernel Submitting Patches

This document mostly focuses on the parts that might be different from the Linux Kernel submitting process.

1. Git clone a sparse repository:

   ```
   git clone git://git.kernel.org/pub/scm/devel/sparse/sparse.git
   ```

2. Coding Style remains the same.

3. Sign off the patch.

   The usage of the Signed-off-by tag is the same as Linux Kernel Sign your work.

   Notice that sparse uses the MIT License.

6.2 TODO

6.2.1 Essential

- SSA is broken by simplify_loads() & branches rewriting/simplification
- attributes of struct, union & enums are ignored (and possibly in other cases too).
- add support for bitwise enums

6.2.2 Documentation

- document the extensions
- document the API
- document the limitations of modifying ptrlists during list walking
- document the data structures
- document flow of data / architecture / code structure

6.2.3 Core

- if a variable has its address taken but in an unreachable BB then its MOD_ADDRESSABLE may be wrong and it won't be SSA converted.
  - let kill_insn() check killing of SYMADDR,
  - add the sym into a list and
  - recalculate the addressability before memops's SSA conversion
- bool_ctype should be split into internal 1-bit / external 8-bit

6.2.4 Testsuite

- there are more than 50 failing tests. They should be fixed (but most are non-trivial to fix).

6.2.5 Misc

- GCC's -Wenum-compare / clangs's -Wenum-conversion -Wassign-enum
- parse *__attribute__*((fallthrough))
- add support for format(printf()) (WIP by Ben Dooks)
- make use of UNDEFs (issues warnings, simplification, . . . ?)
- add a pass to inline small functions during simplification.

6.2.6 Optimization

- the current way of doing CSE uses a lot of time
- add SSA based DCE
- add SSA based PRE
- Add SSA based SCCP
- use better/more systematic use of internal verification framework

6.2.7 IR

- OP_SET should return a bool, always
- add IR instructions for va_arg() & friends
- add a possibility to import of file in "IR assembly"
- dump the symtable
- dump the CFG

6.2.8 LLVM

- fix . . .

6.2.9 Internal backends

- add some basic register allocation
- add a pass to transform 3-addresses code to 2-addresses
- what can be done for x86?

6.2.10 Longer term/to investigate

- better architecture handling than current machine.h + target.c
- attributes are represented as ctypes's alignment, modifiers & contexts but plenty of attributes doesn't fit, for example they need arguments.
  - format(printf, ...),
  - section("...")
  - assume_aligned(alignment[, offsert])
  - error("message"), warning("message")
  - ...
- should support "-Werror=..." ?
- All warning messages should include the option how to disable it. For example:

  "warning: Variable length array is used."

  should be something like:

  "warning: Variable length array is used. (-Wno-vla)"
- ptrlists must not have elements removed while being iterated; this should somehow be enforced.
- having 'struct symbol' used to represent symbols *and* types is quite handy but it also creates lots of problems and complications
- Possible mixup of symbol for a function designator being not a pointer? This seems to make evaluation of function pointers much more complex than needed.
- extend test-inspect to inspect more AST fields.
- extend test-inspect to inspect instructions.

CHAPTER 7

Documentation

7.1 How to write sparse documentation

7.1.1 Introduction

The documentation for Sparse is written in plain text augmented with either reStructuredText (.rst) or MarkDown (.md) markup. These files can be organized hierarchically, allowing a good structuring of the documentation. Sparse uses Sphinx to format this documentation in several formats, like HTML or PDF.

All documentation related files are in the `Documentation/` directory. In this directory you can use `make html` or `make man` to generate the documentation in HTML or manpage format. The generated files can then be found in the `build/` sub-directory.

The root of the documentation is the file `index.rst` which mainly lists the names of the files with real documentation.

7.1.2 Minimal reST cheatsheet

Basic inline markup is:

- `*italic*` gives *italic*
- `**bold**` gives **bold**
- ``` ``mono`` ``` gives `mono`

Headings are created by underlining the title with a punctuation character; it can also be optionally overlined:

```
#############
Major heading
#############

Minor heading
-------------
```

Any punctuation character can be used and the levels are automatically determined from their nesting. However, the convention is to use:

- # with overline for parts

- * with overline for chapters
- = for sections
- – for subsections
- ^ for subsubsections

Lists can be created like this:

```
* this is a bulleted list
* with the second item
  on two lines
* nested lists are supported

        * subitem
        * another subitem

* and here is the fourth item

#. this is an auto-numbered list
#. with two items

1. this is an explicitly numbered list
2. with two items
```

Definition lists are created with a simple indentation, like:

```
term, concept, whatever
        Definition, must be indented and
        continue here.

        It can also have several paragraphs.
```

Literal blocks are introduced with ::, either at the end of the preceding paragraph or on its own line, and indented text:

```
This is a paragraph introducing a literal block::

        This is the literal block.
        It can span several lines.

        It can also consist of several paragraphs.
```

Code examples with syntax highlighting use the *code-block* directive. For example:

```
.. code-block:: c

        int foo(int a)
        {
                return a + 1;
        }
```

will give:

```c
int foo(int a)
{
        return a + 1;
}
```

7.1.3 Autodoc

Sparse source files may contain documentation inside block-comments specifically formatted:

```
///
// Here is some doc
// and here is some more.
```

More precisely, a doc-block begins with a line containing only `///` and continues with lines beginning by `//` followed by either a space, a tab or nothing, the first space after `//` is ignored.

For functions, some additional syntax must be respected inside the block-comment:

```
///
// <mandatory short one-line description>
// <optional blank line>
// @<1st parameter's name>: <description>
// @<2nd parameter's name>: <long description
// <tab>which needs multiple lines>
// @return: <description> (absent for void functions)
// <optional blank line>
// <optional long multi-line description>
int somefunction(void *ptr, int count);
```

Inside the description fields, parameter's names can be referenced by using `@<parameter name>`. A function doc-block must directly precede the function it documents. This function can span multiple lines and can either be a function prototype (ending with `;`) or a function definition.

Some future versions will also allow to document structures, unions, enums, typedefs and variables.

This documentation can be extracted into a .rst document by using the *autodoc* directive:

```
.. c:autodoc:: file.c
```

For example, a doc-block like:

```
///
// increment a value
//
// @val: the value to increment
// @return: the incremented value
//
// This function is to be used to increment a
// value.
//
// It's strongly encouraged to use this
// function instead of open coding a simple
// ``++``.
int inc(int val)
```

will be displayed like this:

int **inc**(int *val*)

Parameters

- **val** – the value to increment

Returns the incremented value

This function is to be used to increment a value.

It's strongly encouraged to use this function instead of open coding a simple ++.

7.1.4 Intermediate Representation

To document the instructions used in the intermediate representation a new domain is defined: 'ir' with a directive:

```
.. op: <OP_NAME>
       <description of OP_NAME>
       ...
```

This is equivalent to using a definition list but with the name also placed in the index (with 'IR instruction' as descriptions).

CHAPTER 8

Release Notes

8.1 Release Notes

8.1.1 v0.6.2 (2020-06-21)

- **add a new tool: sindex - the semantic utility** Sindex is a simple to use cscope-like tool but understanding how symbols are used and which can track struct members.
- add support for GCC's __auto_type
- add support for _Generic
- **fully propagate declarations downward.** For example, it means that code like:

```
static int foo(void);
int foo(void) { return 0; }
```

  now behaves as expected: foo() is effectively static.

- **multi-arch:**
  - allow a single sparse executable to be used for multiple architectures
  - add support for -mcmodel & -f{pic,PIC,pie,PIE}, mainly for RISC-V
  - add new option, –arch=$ARCH, to specify the target architecture
  - move all arch-specific code into separate files (target-$ARCH.c)
  - try to support the various floating-point ABIs on ARM
  - fix wchar_t & wint_t for openbsd
  - add missing predefines for PPC
  - add missing predefines: __amd64 & __amd64__
  - sparc32 on SunOS/Solaris uses 128-bit long double
  - fix wchar_t & wint_t on SunOS/Solaris
  - teach sparse about -fshort-wchar
  - keep cygwin specifics with i386/x86-64 specifics

  - keep BSD & Darwin specifics with i386/x86-64 specifics
  - fix the signedness of plain chars
  - add support for s390 (ILP32)
  - add predefine for __mips__
  - predefine “i386” if needed
  - pre-define __unix__ and friends
  - add necessary defined for sunos-derived systems
  - improved detection of the native OS
- **warnings:**
  - improve diagnostic message about wrong redeclaration
  - conditionally accept { 0 } without warnings
  - add -Wexternal-function-has-definition
  - display the bitfield name in error messages
  - oversized bitfields are now errors
  - add an option to suppress warning ‘no newline at EOF’
  - warn when jumping into statement expressions
  - warn when using undefined labels
  - warn on defined but unused labels
- **attributes:**
  - allows ‘__<attribute-name>__’ for all attributes.
  - improve handling of function attributes
  - separate modifiers into type/declaration
  - add support for attributes ‘unused’ & ‘gnu_inline’
  - simplify parsing of inline/__tls/__visible
  - better handle function-only attributes
  - teach sparse about gnu_inline
  - parse enum attributes and, for now, ignore them
- **cgcc:**
  - use -fshort-char for Cygwin
  - add support for riscv32 & riscv64
  - don’t define __CYGWIN32__ on 64-bit
  - filter-out sparse-specific -msize-long & -msize-llp64
  - use -mfloat-abi=hard for armhf
  - define _BIG_ENDIAN when needed
  - remove definition of _STRING_ARCH_unaligned (defined by glibc)
  - removed unneeded predefines for integers (now defined by sparse)
  - better multi-arch support by using –arch=$ARCH
- **testsuite:**
  - avoid standard includes in the tests

  - fix testcase with non-constant initializer
- **IR**
  - add support for the linearization of builtins
  - generate OP_UNREACH from __builtin_unreachable()
  - add OP_UNREACH after calls to __noreturn functions
- **doc:**
  - do not use obsolete sphinx's AutodocReporter
  - Sphinx's minimal version is now 1.7
  - add basic doc about the type system
  - doc is now accessible as: https://sparse.docs.kernel.org
  - release notes (old and current ones) have been added to the doc
  - now using the sphinx_rtd_theme instead of the classic theme
- **misc:**
  - add support for '-std=c17/c18'
  - simplify testing of which version of the standard is used
  - ensure that typeofs are evaluated before using show_typename()
  - use a single way to expand typeofs
  - various improvements to the 'dissect' tool
  - simplify the parsing of type specifiers
  - improve diagnostic messages concerning bitfields
  - fix premature examination of dereferenced object
  - various fixes for the expansion of constant symbols
  - fix type compatibility of _Atomic types
  - add support for builtin macros with argument
  - add support for __has_feature() & __has_extension()

8.1.2 v0.6.1 (2019-10-14)

It's a small, 74 patches, release containing mainly small fixes and improvements:

- improve build & test support for distros, mainly Debian
- stop warning on externally_visible functions without a prototype
- accept casts of __user/__iomem/... pointers to/from uintptr_t
- fix the underlying type of some enumeration values
- fix a build problem for sparse-llvm by using 'llvm-config –cppflags'
- conditionals (?:) may now be considered as constants if the condition is
- some error messages are now clearer or more coherent
- add missing expansion of compound literals
- improve parsing & checking of asm operands
- add missing expansion of asm operands
- expand some more builtins with constant operands (ffs, clz, ...)

- fix sparsec with recent version of cygwin
- fix crashes with some tools on toplevel asm.

Many thanks to people who have contributed to this release: Uwe Kleine-König, Ramsay Jones, Randy Dunlap, Thomas Weißschuh, Dan Carpenter, Jann Horn, Ben Dooks, Vegard Nossum, Aurelien Aptel, Oliver Hartkopp, Linus Torvalds and Ilya Maximets.

The source code can be found at its usual repository: git://git.kernel.org/pub/scm/devel/sparse/sparse.git v0.6.1

The tarballs are found at: https://www.kernel.org/pub/software/devel/sparse/dist/

8.1.3 v0.6.0 (2018-12-26)

The source code can be found at its usual repository: git://git.kernel.org/pub/scm/devel/sparse/sparse.git v0.6.0

The tarballs are found at: https://www.kernel.org/pub/software/devel/sparse/dist/

Many thanks to people who have contributed to the 888 non-merge patches of this release:

> Ramsay Jones, Randy Dunlap, Uwe Kleine-König, Joey Pabalinas, John Levon, Ben Dooks, Jann Horn, Logan Gunthorpe, Vincenzo Frascino and Tycho Andersen.

Special thanks to Ramsay Jones who has reviewed numerous of my patches, tested many of my series and found many of my typos.

Best wishes for 2019 – Luc Van Oostenryck

The changes since the pre-release (v0.6.0-rc1) are:

- add -Wbitwise-pointer to warn on casts to/from bitwise pointers
- beautify how types are displayed:
- no more <noident>
- no more redundant type specifiers
- remove double space
- some cleanup of the build system:
- only need includedir from llvm-config
- check if sparse-llvm needs libc++
- remove -finline-functions from CFLAGS
- some random cleanup:
- remove unneeded declarations in “compat.h”
- remove unused arg in add_branch()
- allocate BBs after the guards
- remove redundant check of _Bool bitsize
- remove unused regno()
- remove self-assignment of base_type
- some documentation:
- update comment for struct-expression::cast_expression
- fix list formatting in inline doc
- document that identifiers are now OK for address spaces
- add TODO list.

The main changes since the previous release (v0.5.2) are:

- by default, disable warnings about unknown attributes
- no more warnings about sizeof(void) unless -Wpointer-arith is given
- add support for __has_attribute() & __has_builtin()
- many fixes for type evaluation/checking
- the build should be more friendly for distros
- a huge number of testcases have been added to the testsuite
- the handling of cast instructions has been completely revamped
- the SSA conversion has been is now corrected and has been rewritten with a variant of the classical algorithm
- documentation is now handled with Sphinx and inline doc is extracted from the code.
- online documentation can be found at https://sparse-doc.readthedocs.io/en/master/

A more complete list of changes is:

- add predefined macros for __INTMAX_TYPE__, __INT_MAX__, ...
- prepare to identify & display the address spaces by name
- fix linearization of non-constant switch-cases
- manpage: update maintainer info
- manpage: add AUTHORS section
- fixes for -dD
- add support for -dM
- remove more complex phi-nodes
- fix linearization/SSA when missing a return
- fix linearization/SSA of (nested) logical expressions
- fix linearization of unreachable switch + label
- add support for __has_attribute()
- consolidate instruction's properties into an opcode table
- fix: do not optimize away accesses to volatile bitfields
- support mode(__pointer__) and mode(__byte__)
- do 'classical' SSA conversion (via the iterated dominance frontier).
- fix buggy recursion in kill_dead_stores()
- kill dead stores again after memops simplification is done.
- simplify TRUNC((x & M') | y, N)
- simplify AND(SHIFT(a | b, S), M)
- simplify TRUNC(SHIFT(a | b, S), N)
- add simplification of TRUNC(TRUNC((x))
- add simplification of SHL(LSR(x), S), S)
- generate integer-wide OP_ADD & OP_SUB in linearize_inc_dec()
- simplify mask instructions & bitfield accesses
- fix a few bugs related to the linearization of logical expressions
- simplify those logical expressions.

- add optimized version of some list operations
- some simplifications of OP_SETNE & OP_SETEQ with 0 and 1
- several simplifications involving casts and/or bitfields
- give a correct & sensible warning on negative or over-sized shifts.
- add conservative simplification of such shifts.
- do not optimize the meaningless shift:
  - any shift with a negative count
  - OP_ASRs with an over-sized shift count.
- try to give a correct negative/too-big error message during simplification.
- simplify chains of shifts.
- simplify ZEXT + ASR into ZEXT + LSR
- cse: try the next pair instead of keeping the first instruction
- cse: compare casts only by kind a size, not by C type.
- optimize away OP_UTPTR & OP_PTRTU which are nops.
- cleanup of list walking macros:
  - make untagged pointers the normal case
  - use structurally equivalent struct for all pointer lists to avoid needless casting to and fro struct ptrlist
  - simplify PREPARE/NEXT/RESET logic by using common PTR_NEXT()
- add validation of the IR.
- improve expansion of builtin dynamic macros (__FILE__, . . .)
- add support for __INCLUDE_LEVEL__ & __BASE_FILE__
- improve generation of predefined macros
- add support for builtins doing overflow checking.
- add support for the __has_builtin() macro.
- improve Sphinx doc for IR instructions
  - have those instructions in the index
  - have a nicer presentation of the generated doc thanks to not having to use level-4 headings anymore
- fixes & improvements to the testsuite; mainly:
  - allow to run the testsuite on all the tests of a subdir
  - teach 'format' to directly append to the testcase
  - validate the 'check-. . . ' tags

Shortlog

Ben Dooks (1):

- tokenize: check show_string() for NULL pointer

Jann Horn (1):

- fix accesses through incorrect union members

Joey Pabalinas (2):

- doc: copy-edit text related to applying sizeof to a _Bool

- sparse: add -Wpointer-arith flag to toggle sizeof(void) warnings

John Levon (2):

- Ignore #ident directives
- Conditionalize 'warning: non-ANSI function ...'

Logan Gunthorpe (1):

- add __builtin functions for isinf_sign, isfinite and isnan

Luc Van Oostenryck (857):

- use long for all mem stats
- diet: use smaller LIST_NODE_NR (29 -> 13)
- diet: remove unused struct scope::token
- diet: remove unused struct symbol::arg_count
- option: add helper to parse/match command line options
- option: rename 'struct warning' to 'struct flag'
- option: let handle_simple_switch() handle an array of flags
- option: extract OPTION_NUMERIC() from handle_switch_fmemcpy_max_count()
- option: add support for options with 'zero is infinity'
- option: add support for '-<some-option>=unlimited'
- option: use OPTION_NUMERIC() for handle_switch_fmemcpy_max_count()
- option: constify match_option()
- option: handle switches by table
- dump-ir: add defines for the compilation passes
- testsuite: 'echo -n' may not be interpreted as '-n'
- testsuite: allow to test a few cases at once
- testsuite: move verbose() & error()
- testsuite: better message for pattern nbr checking
- testsuite: better message for pattern absence/presence
- use shorter name for constexpr tests
- testsuite: new eq/min/max syntax for pattern checking
- testsuite: obsolete old pattern checking syntax
- testsuite: convert to the new pattern syntax
- use a specific struct for asm operands
- fix: missing evaluate with '-include' : add testcase
- fix: missing evaluate with '-include'
- fix test case kill-phi-ttsb
- add test case for incomplete type
- add test case for bad return type
- diet: remove unused struct symbol::value
- cclass: char is wide enough
- cclass: cleanup

- remove prototype extern int is_ptr_type()
- remove prototype for nonexistent examine_simple_symbol_type()
- graph: do not scan removed instructions
- build: fix effectiveness of generated dependencies
- build: remove unused support for pkgconfig
- cgcc: teach cgcc about freebsd & netbsd
- testsuite: clearer result summary
- testsuite: check error messages first
- testsuite: saner handling of 'must_fail'
- testsuite: allow to parse several options
- testsuite: add support for -q|–quiet
- testsuite: add support for -a|–abort
- testsuite: get options from env too
- testsuite: allow –format & –single
- testsuite: remove useless selftest
- testsuite: remove useless test-be.c
- testsuite: extract disable()
- testsuite: simplify documentation
- testsuite: allow arch-specific tests
- testsuite: save screen real estate
- testsuite: add a blank line before format
- testsuite: 'quiet' must be initialized earlier
- testsuite: move up arg_file()
- testsuite: make do_format() more self-contained
- testsuite: format: saner defaults handling
- testsuite: format: strip .c from default name
- testsuite: add support for 'format -f'
- testsuite: add support for 'format -l'
- remove never-used MOD_TYPEDEF
- MOD_ACCESSED is not a type modifier ...
- reorganize the definition of the modifiers
- remove redundancy in MOD_STORAGE
- define MOD_QUALIFIER for (MOD_CONST | MOD_VOLATILE)
- associate MOD_RESTRICT with restrict-qualified variables
- add support for C11's _Atomic as type qualifier
- build: use '-objs' instead of '_EXTRA_DEPS'
- build: use '-ldlibs' instead of '_EXTRA_OBJS'
- build: allow target-specific CFLAGS, CPPFLAGS, LDFLAGS & LDLIBS
- build: allow CFLAGS & friends from command line

- build: avoid rule-specific CFLAGS
- build: use $LIBS directly in the dependency list
- build: no need to use wildcards for generated dependencies
- build: reuse rule for ALL_OBJS
- build: CHECKER_FLAGS=-Wno-vla for all targets
- build: move tests near their use
- build: add note about overwritable vars
- build: remove references to nonexistent pre-process.h
- build: move clean & clean-check together
- build: make clean targets quieter
- build: remove rule for shared lib, it's unused
- build: normalize rules
- build: remove the dist rule since unused
- build: use one line per item
- build: allow the name 'local.mk' to be configurable via the environment
- build: use standard rules for install
- build: remove unused QUIET_INST_SH
- build: let quiet commands use less indentation
- build: simplify quiet commands
- build: simplify clean pattern
- build: add *.o to clean-check pattern
- build: avoid foreach
- build: reorg & add comment
- build: use a single space before assignments
- build: add rule to run a single test
- build: let -fno-strict-aliasing be a mandatory flag
- volatile loads are side-effects too
- define MOD_ACCESS for (MOD_ASSIGNED | MOD_ADDRESSABLE)
- fix 'simplification' of float-to-int casts
- fix description setval & symaddr
- flush stdout when warning
- add test case for bogus volatile simplification
- fix: volatile stores must not be simplified
- dump-ir: add testcase for option parsing corner case
- dump-ir: allow to specify the passes to execute via cli's options
- dump-ir: activate/deactivate pass 'mem2reg'
- dump-ir: allow to skip the optimization pass(es)
- dump-ir: saner use of fdump_linearize
- dump-ir: rename -fdump-linearize to -fdump-ir

- dump-ir: make it more flexible
- dump-ir: activate -fdump-ir=mem2reg
- add test case for using multiple input files
- add test case for VLA sizeof
- add test case for memory to register problem
- add test case for conditionally undefined var
- add test case for incomplete type
- add test case bitfields in K&R decl
- add test case storage specifier in struct member
- add test case using sizeof on incomplete type
- add test case for bad layout of bool in bitfields
- add test case for missed overflow detection
- add test cases for canonicalization of add/sub chains
- add test case for compound literals
- add testcase for __builtin_unreachable()
- add test cases for canonicalization of mul chains
- add test case for pre-processor extra tokens warning
- add testcase for return & inline
- add test cases for simplification of equivalent to ‘x == 0’ or ‘x != 0’
- add test case for superfluous cast with volatiles
- add testcase for mem2reg/SSA conversion
- add test cases for canonicalization of boolean expressions
- add test case for missing conversion to select
- show OP_PHI without VOID
- don’t output value of anonymous symbol’s pointer
- add table to “negate” some opcode
- use opcode table for compare_opcode()
- canonicalize binops before simplification
- canonicalize compare instructions
- add is_signed_type()
- fix usage of inlined calls
- inlined calls should not block BB packing
- give a type to all function arguments
- give a type to OP_PHISOURCEs
- give a type to OP_SELs, always
- give a type to OP_SWITCHs
- add doc about sparse’s instructions/IR
- add support for wider type in switch-case
- llvm: remove unneeded arg ‘module’

- llvm: remove unneeded ‘generation’
- llvm: remove unneeded function::type
- llvm: reduce scope of ‘bb_nr’
- llvm: use pseudo_list_size() instead of open coding it
- llvm: give arguments a name
- llvm: give a name to call’s return values
- llvm: avoid useless temp variable
- llvm: extract get_sym_value() from pseudo_to_value()
- llvm: fix test of floating-point type
- llvm: fix translation of PSEUDO_VALs into a ValueRefs
- llvm: fix output_op_store() which modify its operand
- llvm: fix output_op_[ptr]cast()
- llvm: add test cases for symbol’s address
- llvm: add test cases for pointers passed as argument
- llvm: add test cases for arrays passed as argument
- llvm: add test cases for degenerated pointers
- llvm: add support for OP_NEG
- llvm: add support for OP_SETVAL with floats
- llvm: add support for OP_SETVAL with labels
- llvm: ignore OP_INLINED_CALL
- llvm: fix pointer/float mixup in comparisons
- llvm: fix type in comparison with an address constant
- llvm: give correct type to binops
- llvm: adjust OP_RET’s type
- llvm: variadic functions are not being marked as such
- llvm: fix type of switch constants
- llvm: make pseudo_name() more flexible
- llvm: give a name to all values
- llvm: add support for OP_SWITCH with a range
- llvm: fix OP_SWITCH has no target
- llvm: make value_to_pvalue() more flexible
- llvm: make value_to_ivalue() more flexible
- llvm: add test case pointer compare with cast
- llvm: let pseudo_to_value() directly use the type
- llvm: add small script to test LLVM generated bytecode
- llvm: add testcase for calling variadic functions
- llvm: fix variadic calls with constants
- llvm: take care of degenerated rvalues
- llvm: fix mutating function pointer

- llvm: fix mutated OP_RET
- llvm: fix mutated OP_SEL
- llvm: fix mutated OP_SWITCH
- llvm: fix mutated OP_PHISOURCE
- llvm: fix mutated OP_[PTR]CAST
- llvm: add support for restricted types
- llvm: fix get value from initialized symbol
- llvm: fix get value from non-anonymous symbol
- llvm: fix type of bitfields
- llvm: add support for OP_FPCAST
- llvm: add support for cast from floats
- llvm: cleanup of output_[ptr]cast()
- llvm: fix creation of sparsec's tmp files
- llvm: give names easier to debug
- llvm: gracefully catch impossible type/value
- llvm: warn instead of assert on global inits
- llvm: add support for float initializer
- llvm: only compare void pointers
- fix linearize_inc_dec() with floats
- add test case for boolean negation on float
- fix support of floating-point compare
- add support of floating-point specific arithmetic ops
- testsuite: fix: remove unneeded './' before commands
- fix: build sparse-llvm on i686s too.
- add more testcases for using addresses in conditionals
- add testcases linearization of degenerated arrays/functions
- fix: add missing degenerate() for logical not
- testsuite: make the '%.t' rule depends on PROGRAMS too
- testsuite: fix a few more incorrect check-commands
- testsuite: convert to the new pattern syntax
- testsuite: remove old ugly pattern syntax
- testsuite: move verbose/error() before get_tag_value()
- testsuite: add & use warning()
- testsuite: reset 'quiet' at the start of each testcase
- testsuite: fix invalid 'check-...' tags
- testsuite: validate the 'check-...' tags
- testsuite: early return in getopt loop
- testsuite: move do_test_suite out of the getopt loop
- testsuite: move no-arg out of the getopt loop

- testsuite: change do_usage text
- testsuite: allow to test only a subdir
- testsuite: default to shift in the getopt loop
- testsuite: add support for 'format -a'
- add testcase for 'sparse -D M...'
- fix: accept 'sparse -D M...'
- testsuite: add test case for quoting of command's arguments
- testsuite: process extra options without exec
- testsuite: respect command line's quotes & whitespaces
- testsuite: allow default args from environment for test commands
- add test case for space within command line
- fix: spaces in macro definition on the command line
- add testcases for unexamined base type
- fix: evaluate_dereference() unexamined base type
- add testcases for the linearization of calls
- simplify linearize_call_expression()
- fix linearize (*fun)()
- add testcases for multiple deref of calls
- avoid unneeded alloc on error path
- dereference of a function is a no-op
- add testcase for constant bitfield dereference
- fix expansion of constant bitfield dereference
- add testcase for CSE of floating-point compares
- fix: restore CSE on floating-point compares
- llvm: fix: previous function ref MUST be reused
- llvm: use LLVMModuleRef for get_sym_value()
- llvm: add declares for function prototypes
- testcases: add missing return statements
- warn on empty parenthesized expressions
- fix crash on bad expression in linearize_switch()
- llvm: simplify emit of null pointers
- llvm: default init of arrays & structs
- add more testcases for function designator dereference
- add testcases for type comparison
- fix implicit size of unsized arrays
- let handle_switches() also handle reverse logic
- add support for '-f[no-][un]signed-char'
- fix: dereference null-type
- teach sparse about '-fmax-warnings'

- give a type to builtin functions
- ctags: avoid null deref
- cleanup: make some functions static
- cleanup: remove unused & obsolete symbol_is_typename()
- cleanup: remove unused delete_last_basic_block()
- cleanup: remove declaration of unused merge_phi_sources
- add OP_SETFVAL
- CSE: support CSE of floating-point literal
- doc: fix manpage formatting
- report type & size on non-power-of-2 pointer subtraction
- remove warning "call with no type"
- add testcases for duplicated warning about invalid types
- fix error in bad conditional
- early return if null ctype in evaluate_conditional()
- add helper: valid_type()
- use valid_type to avoid to warn twice on conditionals
- add helpers: valid_expr_type() & valid_subexpr_type()
- do not report bad types twice
- always evaluate both operands
- fix examination of bad typeof
- extract extract eval_insn() from simplify_constant_binop()
- add testcase of dead dominator
- fix dead dominator
- fix missing checks for deleted instructions
- add testcase for bad killing of dominated stores
- add helper for pseudo's user-list's size
- add helper: has_users()
- use has_users() in dead_insn() too
- let kill_instruction() report if changes were made
- add testcases for converted loads
- fix killing of converted loads
- fix usage of deadborn loads
- kill dead loads
- kill dead stores when simplifying symbols
- By default disable the warning flag '-Wunknown-attribute'
- no repetition in unknown attribute warning message
- cleanup: remove unused 'struct pseudo_ptr_list'
- llvm: use list_size() to count the numbers of arguments
- llvm: initialize at declaration time

- show_pseudo(): protect against NULL ->sym
- use show_pseudo() for OP_SYMADDR's symbol
- let struct access_data use a single type
- rename base_type() to bitfield_base_type()
- builtin: add ctype for const {void,char} *
- builtin: make builtins more builtin
- builtin: extract eval_args() from arguments_choose()
- builtin: add typechecking of isnan(), isinf(), ...
- builtin: add testcases for expansion of special FP constants
- builtin: add testcases for expansion of FP classification
- unsigned multiplication is also associative
- no need for signed & unsigned multiplication
- use '%lld' for printing long longs
- build: add -MP for generated dependencies
- build: use -MMD for generated dependencies
- ban use of 'true' or 'false'
- 'amd64' is also ok for sparse-llvm
- testsuite: fix typo with 'test-suite format -a'
- rename variable 'optimize' to 'optimize_level'
- move the optimization loop in its own file
- cse: untangle simplification & hashing
- extract cse_eliminate() from cleanup_and_cse()
- expose interface to CSE
- move liveness interface to its own header
- move inner optimization loop into optimize.c
- move the inner optimization loop into the main loop
- remove unneeded cast in calls to free_ptr_list()
- testsuite: add testcase for some random crash
- testsuite: add testcase about CSE problem
- IR: fix typo in IR doc
- IR: remove now unused OP_LNOP & OP_SNOP
- IR: remove never-generated instructions
- IR: let .cond unionize with .src and not .target
- IR: let OP_COMPUTEGOTO use .src instead of .target
- llvm: normalize sparse-llvm-dis' output
- llvm: fix typo for constant addresses
- testsuite: fix problem with double-escaping in patterns
- add a field 'tainted' to struct instruction
- taint: let check_access() warn just once

- fix address_taken()
- fix symbol cleanup
- cleanup deadborn phi-sources
- optim: add some more optimization tests
- optim: add testcase for internal infinite loop
- optim: add timeout for infinite optim loop tests
- optim: kill unreachable BBS after CFG simplification
- optim: no need to kill_unreachable_bbs() after main loop
- optim: fix optimization loop's condition
- optim: pack bb must set REPEAT_CFG
- optim: load simplification should repeat optimization
- optim: fix REPEAT_CFG_CLEANUP
- add an helper to test the value of a pseudo against zero
- optim: simplify null select
- make remove_usage() more generic
- add remove_use()
- show_label: add (and use) show_label()
- extract alloc_phisrc() from alloc_phi()
- small code reorg of add_store()
- alloc: add missing #include "compat.h"
- defer initialization of bb::context
- fix-return: remove special case for single return
- avoid deadborn loads & stores
- doc: options.md is for development
- doc: document the debug flags
- fix missing handling of OP_FNEG
- graph: do not use insn->symbol for memops
- use -Wpointer-arith for tests
- default to LP64 for all and only for 64 bit ABIs
- fix alignment of 64 bit integers on LLP64
- add testcases for verifying ABI's integer size & align
- use an enum for ARCH_LP32 & friends
- add a flag -mx32 ILP32 env on 64 bit archs
- add testcase for enum / int type difference
- add testcase for array size type difference
- add testcase for typedef redefinition
- export check_duplicates()
- fix: warn on typedef redefinition
- do not to ignore old preprocessor testcases

- use also __x86_64 when __x86_64__ is used
- build: use a variable for $(shell uname -m)
- build: use 'filter' to do pattern matching inside the Makefile
- build: disable LLVM on x86-64-x32
- let cgcc use sparse's predefines for i386 & x86-64
- build: use –dirty with 'git describe'
- fix build on Hurd which doesn't define PATH_MAX
- testsuite: add check-cp-if
- testsuite: add check-assert
- teach sparse about _Floatn and _Floatnx
- add test case bug expand union
- alloc: check if size is too big
- fix: don't dump pointer value in error message
- fix missing checks for deleted instructions
- fix comment about PSEUDO_SYM usage
- fix: remove usage when killing symaddr (part 1)
- fix: remove usage when killing symaddr (part 2)
- OP_SYMADDR is simply an unop
- use function-like syntax for __range__
- increment the version number suffix it with -dev
- doc: fix markdown syntax
- doc: fix headings
- doc: add minimal support for sphinx-doc
- doc: add logo
- doc: automatically set the copyright date
- doc: automatically get the version
- doc: set primary domain to C
- doc: allow .md with py3-sphinx
- doc: move sparse.txt to markdown and rename it
- doc: the testsuite doc in reST
- doc: format dev-options.md as a man page
- doc: use reST for manpages
- api: move evaluate interface to its own header file
- doc: add structured doc to ptrlist.c
- autodoc: extract doc from the C files
- autodoc: convert extracted doc to reST
- autodoc: add a sphinx c:autodoc directive for the extracted doc
- autodoc: add doc from ptrlist.c
- autodoc: add markup to argument's references

- autodoc: doc the doc
- autodoc: by default disable syntax highlighting
- autodoc: add a small cheatsheet for reST markup
- autodoc: support muti-line param & return descriptions
- autodoc: document a few more APIs to test multiline
- autodoc: add autodoc tests in the testsuite
- doc: convert IR.md to reST
- doc: add sphinx domain for IR instruction indexation
- add helper for new parsing errors: unexpected()
- context: fix parsing of attribute 'context'
- context: __context__(...) expect a constant expression
- context: fix crashes while parsing '__context__;' or '__context__(;'
- context: stricter syntax for __context__ statement
- context: extra warning for __context__() & friends
- label: add testcase for label redefinition
- label: avoid multiple definitions
- vla-sizeof: add test cases
- vla-sizeof: add support for sizeof of VLAs
- fix typing of __builtin_expect()
- fix crash on 'goto <reserved word>'
- give a position to end-of-input
- avoid multiple error message after parsing error
- add test for integer-const-expr-ness
- dyn-macro: add testcase for __LINE__ & friends
- dyn-macro: use a table to expand __DATE__, __FILE__, ...
- dyn-macro: add support for __INCLUDE_LEVEL__
- dyn-macro: add real support for __BASE_FILE__
- utils: add xmemdup() & xstrdup()
- utils: convert alloc + copy to {mem,str}dup_alloc()
- builtin: add testcase for builtin macro expansion
- builtin: extract do_define() from do_handle_define()
- builtin: add builtin types {u,}{int,long,long}_ptr_ctype
- builtin: declare __builtin_[us]{add,sub,mul}{,l,ll}_overflow()
- builtin: rename arguments_choose() to args_triadic()
- builtin: add support for __builtin_{add,sub,mul}_overflow(), ...
- extract replace_with_bool() from replace_with_defined()
- builtin: add support for __has_builtin()
- builtin: add predefine()
- builtin: directly predefine builtin macros

- builtin: consolidate predefined_macros()
- doc: API before IR
- builtin: switch calling order of predefined_macros() & friends
- builtin: merge declare_builtin_function() with declare_builtins()
- teach sparse about -m16
- ptrlist: specialize __add_ptr_list() for tag/notag
- ptrlist: remove now unneeded add_ptr_list_notag()
- ptrlist: add helper PTR_UNTAG()
- ptrlist: rename PTR_ENTRY() to PTR_ENTRY_UNTAG()
- ptrlist: make explicit when tagged pointers are used.
- ptrlist: let FOR_EACH_PTR() ignore tags
- utils: add xasprintf() & xvasprintf()
- add support for -fdiagnostic-prefix[=prefix]
- doc: add doc for the -vcompound flag
- testsuite: fix missing return
- keep the debug flags alphabetically sorted
- testsuite: allow extra/default options to test commands
- ir-validate: add framework for IR validation
- ir-validate: validate pseudo's defining instruction
- ir-validate: add validation of (nbr of) phi operands
- ir-validate: add more validation points
- sparsec: simplify & portable use of mktemp
- add predefines for __INT_WIDTH__ & friends
- ptrlist: remove the now unneeded FOR_EACH_PTR_NOTAG()
- ptrlist: let {first,last}_ptr_list() return the raw pointer
- ptrlist: let sort_list() use the raw pointer
- ptrlist: let all pointer lists have the same parameterized structure
- ptrlist: when possible use the real type of the list
- ptrlist: remove now unneeded CHECK_TYPE()
- ptrlist: make add_ptr_list() more readable
- ptrlist: make free_ptr_list() more readable
- ptrlist: remove some unneeded arg from internal macros.
- ptrlist: remove extra ident level
- ptrlist: simplify loop nesting
- ptrlist: simplify DO_NEXT
- ptrlist: simplify PREPARE/NEXT
- ptrlist: shorter continuated lines
- ptrlist: remove ptr_list_empty()
- ptrlist: make {first,last}_ptr_list() out-of-line functions

- ptrlist: move semi-private prototypes close to their user
- ptrlist: use VRFY_PTR_LIST() for sanity check
- ptrlist: keep declaration of head-list-nr together
- ptrlist: make clear what is API and what is implementation.
- ptrlist: move DO_SPLIT() into DO_INSERT_CURRENT()
- ptrlist: add missing doc for some functions
- add testcase for bad fpcast simplification
- fix bad fpcast simplification
- avoid useless deref in simplify_cond_branch()
- new helper: replace_pseudo()
- remove unused arg in simplify_cond_branch()
- add_uniop() should take a type, not an expression
- rename add_uniop() to add_unop()
- add missing entry for OP_FNEG in kill_insn() & validate_insn()
- ir: define an OP_.. range for unops
- ir: case OP_UNOP ... OP_UNOP_END
- cast: reorg testcases related to casts
- cast: add testcase for bad implicit casts to struct/union
- cast: add testcase for cast to bad typeof
- cast: add tests for warnings issued by sparse -v
- cast: rename evaluate_cast()'s vars with slightly more meaningful names
- cast: force_cast are OK on non-scalar values
- cast: prepare finer grained cast instructions
- cast: specialize FPCAST into [USF]CVTF
- cast: handle NO-OP casts
- cast: specialize floats to integer conversion
- cast: specialize casts from unsigned to pointers
- cast: make [u]intptr_ctype alias of [s]size_t_ctype
- cast: make pointer casts always size preserving
- cast: temporary simplify handling cast to/from void*
- cast: specialize cast from pointers
- cast: add support for -Wpointer-to-int-cast
- cast: make casts from pointer always size preserving
- cast: specialize integer casts
- cast: accept null casts
- cast: do not try to linearize illegal casts
- cse: add testcase for missed opportunity
- new helper: def_opcode()
- cast: simplify simplify_cast()

- cast: merge simplification of constant casts with constant unops
- cast: prepare for more cast simplifications
- cast: keep instruction sizes consistent
- cse: move to next comparable instruction
- cast: simplify TRUNC + ZEXT to AND
- add simple testcases for internal infinite loops
- simplify 'x | ~0' to '~0'
- simplify 'x & ~0' to 'x'
- simplify 'x ^ ~0' to '~x'
- bool: add testcase for bool simplification
- bool: fix add missing check in simplify_seteq_setne()
- bool: simplify ZEXT in bool -> int -> bool
- bool: fix missing boolean context for floats
- bool: generate plain OP_{AND,OR} instead of OP_{AND,OR}_BOOL
- bool: remove OP_{AND,OR}_BOOL instructions
- cast: reorganize testcases for cast optimization
- cast: optimize away casts to/from pointers
- cse: let equivalent casts hash & compare identically
- fix killing OP_SWITCH
- fix: remove dead OP_{SETVAL,SETFVAL,SLICE}
- kds: add testcases for kill_dead_stores()
- kds: add explanation to kill_dead_stores()
- kds: rename kill_dead_stores() to kill_dead_stores_bb()
- kds: add interface for kill_dead_stores()
- kds: kill dead stores after memops simplification
- kds: shortcut for kill_dead_stores()
- kds: fix recursion in kill_dead_stores_bb()
- kds: clarify kill_dead_stores_bb()
- testsuite: reorganize tests for compound literals
- testsuite: add a few more tests catching quadratic behaviour
- testsuite: improve mem2reg testcases
- testsuite: remove useless test for loop-linearization
- graph: build the CFG reverse postorder traversal
- graph: add debugging for (reverse) postorder traversal
- dom: calculate the dominance tree
- dom: add some debugging for the dominance tree
- dom: add support for dominance queries
- dom: build the domtree before optimization
- dom: use domtree for bb_dominates()

- sset: add implementation of sparse sets
- idf: compute the iterated dominance frontier
- idf: add test/debug/example
- add new helper: is_integral_type()
- add PSEUDO_UNDEF & undef_pseudo()
- add insert_phi_node()
- ptrmap: core implementation
- ptrmap: add type-safe interface
- ssa: phase 1: phi-nodes placement
- ssa: phase 2: rename load & stores
- ssa: phase 3: rename phi-nodes
- ssa: activate the new SSA conversion
- ssa: remove unused simplify_symbol_usage()
- ssa: phi worklist
- remove unused finish_address_gen()
- remove unused struct access_data::pos
- no need to assign ad->type for EXPR_POS
- remove obsolete comment: /* Dummy pseudo allocator */
- big-shift: add test for shifts with bad count
- big-shift: mark out-of-range OP_{ASR,LSR,SHL} as tainted
- big-shift: do not evaluate negative or over-sized shifts
- big-shift: don't take the modulo at expand time
- big-shift: move the check into check_shift_count()
- big-shift: use the base type for shift-too-big warning
- big-shift: also check shift count of shift-assignment
- big-shift: do not simplify over-sized OP_ASR to zero
- big-shift: reorder the tests in simplify_asr()
- big-shift: reuse simplify_asr() for LSR & SHL
- big-shift: simplify over-sized OP_LSRs
- big-shift: simplify over-sized OP_SHLs
- big-shift: use the type width for too big shift
- big-shift: fix warning message for negative shift count
- big-shift: fix evaluation of shift-assign
- big-shift: do not truncate the count when checking it
- big-shift: add -Wshift-count-{negative,overflow}
- extract nbr_users() from unssa.c
- add testcases for casts & bitfield insertion/extraction
- bitfield: extract linearize_bitfield_extract()
- bitfield: extract linearize_bitfield_insert()

- cast: simplify [SZ]EXT + TRUNC to original size
- cast: simplify [SZ]EXT + TRUNC to a smaller/greater size
- cast: fix shift signedness in cast simplification
- cast: do not compare sizes, test the opcode
- cast: use a switch to handle TRUNC(AND(x,M),N) in simplify_cast()
- cast: preserve the sizes of TRUNC(AND(x,M),N)
- cast: simplify [ZS]EXT(AND(x,M),N)
- cast: simplify AND(ZEXT(x,M),N)
- cast: simplify SEXT(SEXT(x,N),N')
- cast: simplify ZEXT(ZEXT(x,N),N')
- cast: simplify SEXT(ZEXT(x,N),N')
- bits: add helpers for zero & sign-extension
- big-shift: add testcases for simplification of over-sized shifts
- big-shift: add testcases for simplification of negative shifts
- big-shift: move shift count check in a separate function
- big-shift: fix warning message for negative or over-sized shifts
- big-shift: do not optimize negative shifts
- big-shift: do not optimize over-sized ASRs
- use "%Le" to display floats
- add copy_ptr_list()
- testcases: add testcase for missing detection of out-of-bound stores
- testcases: missing evaluation of side effects in typeof(VLA)
- kill dead OP_FADD & friends
- add ptr_list_empty()
- add ptr_list_multiple()
- add lookup_ptr_list_entry()
- shift: simplify LSR(LSR(x,N),N') & friends
- shift: simplify ASR(LSR(x,N),N')
- shift: avoid simplification of ASR(LSR(x,0),N)
- shift: simplify ASR(ZEXT(X, N), C)
- testcase for SET{EQ,NE}([SZ]EXT(x, N),{0,1})'s simplification
- cleanup of simplify_seteq_setne(): remove tmp vars
- simplify SET{EQ,NE}(ZEXT(x, N),{0,1})
- simplify SET{EQ,NE}(SEXT(x, N),{0,1})
- simplify 'x != 0' or 'x == 1' to 'x'
- add testcase for linearize_logical()
- fix size corruption when simplifying 'x != 0' to 'x'
- protect add_convert_to_bool() against bad types / invalid expressions
- conditional branches can't accept arbitrary expressions

- fix linearize_conditional() for logical ops
- expand linearize_conditional() into linearize_logical()
- simplify linearize_logical()
- simplify SETNE(AND(X,1),0) to AND(X,1)
- simplify SETNE(TRUNC(x,N),{0,1})
- simplify ZEXT(SETCC(x,y), N)
- simplify SEXT(SETCC(x,y), N)
- simplify TRUNC(SETCC(x,y), N)
- simplify AND(SETCC(x,y), M)
- boolean conversion of boolean value is a no-op
- cast: fix warning position in cast_pseudo()
- limit the mask used for bitfield insertion
- expand linearize_position() into linearize_initializer()
- put back the bitfield base type into struct access_data
- fix instruction size & type in linearize_inc_dec()
- fix bad indentation in linearize_inc_dec()
- avoid infinite simplification loops of the second kind
- optim: add a few more testcases for shift & mask
- use multi_users() instead on nbr_users()
- reorg code for shift-shift simplification
- simplify ((x & M') | y) & M into (y & M) when (M' & M) == 0
- simplify ((x & M) | y) >> S to (y >> S) when (M >> S) == 0
- simplify (x << S) >> S into x & (-1 >> S)
- simplify (x & M) >> S to (x >> S) & (M >> S)
- rename testcase for ((x << S) >> S) simplification
- add testcase for ((x >> S) << S) simplification
- add testcase for TRUNC(TRUNC(x)) simplification
- simpler guard in LSR-SHL simplification
- reorganize shift-shift simplification
- simplify ((x >> S) << S)
- reorganize simplification of ZEXT(TRUNC(x))
- simplify TRUNC(TRUNC(x))
- doc: simplify the creation of the viewlist
- doc: automatically insert the blank line for lists
- doc: convert existing simplify.c doc into ReST autodoc
- doc: reword doc for replace_pseudo()
- doc: add doc for simplification notation
- add testcase for (((x & M') | (y & M')) & M)
- add testcases for bitfield & AND/OR simplification

- unify simplify_lsr_or() & simplify_and_or_mask()
- add simplify_mask_or()
- use better names for simplify_mask_or_and() vars
- document simplify_mask_or() & simplify_mask_or_and()
- switch return order in simplify_mask_or_and()
- allow simplification of OP(((x & y) | (a & M')), K)
- move opcode test inside simplify_mask_or_and()
- simplify OP(((x & M') | y), K) when (M' & M) == M
- simplify OP(((x & M') | y), K) when (M' & M) != M'
- simplify OP((x | C), K) when (C & M) == 0
- simplify OP((x | C), K) when (C & M) == M
- simplify OP((x | C), K) when (C & M) != C
- simplify SHL((x & M') | y, S)
- add testcases for {LSR,SHL}(AND(x, M), S) with shared AND(x, M)
- use an intermediate mask in simplify_shift()
- simplify ((x & M) >> S) when (M >> S) == 0
- simplify ((x & M) >> S) when (M >> S) == (-1 >> S)
- simplify ((x & M) << S) when (M << S) == 0
- simplify ((x & M) << S) when (M << S) == (-1 << S)
- simplify TRUNC((x & M') | y, N)
- doc: extend simplification notation
- prepare simplification of MASK(SHIFT(a | b, S), M)
- simplify AND(SHIFT(a | b, S), M)
- simplify TRUNC(SHIFT(a | b, S), N)
- mode keywords don't need MOD_{CHAR,LONG,...}
- add support for mode __pointer__
- add support for mode __byte__
- add a testcase for enum using a mode
- remove superfluous newline in 'unknown mode attribute' error message
- testsuite: remove useless test for loop-linearization
- symaddr: s/insn->symbol/insn->src/
- add testcase for accesses to volatile bitfields
- split memops from unops
- add a flag for volatile memops
- fix: do not optimize away accesses to volatile bitfields
- opcode: centralize opcode definition
- opcode: add arity info
- opcode: add OPF_TARGET
- add a function to remove deadborn instructions

- fix missing declarations
- has-attr: add testcase for __has_attribute()
- has-attr: move 'mode' next to '__mode__'
- has-attr: add __designated_init__ & transparent_union
- has-attr: add support for __has_attribute()
- ir-validate: add validation branch to dead BB
- ir-validate: ignore dead phis
- ir-validate: validate return value
- add testcase for unreachable label in switch
- fix linearization of unreachable switch (with reachable label).
- move DEF_OPCODE() to header file
- trivial-phi: add testcase for unneeded trivial phi-nodes
- trivial-phi: make clean_up_phi() more sequential
- trivial-phi: extract trivial_phi() from clean_up_phi()
- trivial-phi: early return
- trivial-phi: use a temp var for the real source
- trivial-phi: directly return the unique value
- trivial-phi: remove more complex trivial phi-nodes
- stricter warning for explicit cast to ulong
- add linearization as a pass
- add testcases for missing return in last block
- use a temp var for function's upper-level statement
- topasm: top-level asm is special
- specialize linearize_compound_statement()
- there is always an active BB after linearize_fn_statement()
- the return BB is never terminated
- extract add_return() from linearize_return()
- use UNDEF for missing returns
- use a temp var for the return type/symbol
- return nothing only in void functions
- add testcases for wrong ordering in phi-nodes
- fix ordering of phi-node operand
- add tests for nested logical expr
- fix linearization of nested logical expr
- add testcase for non-constant switch-case
- fix linearization of non-constant switch-cases
- test: make test Waddress-space-strict succeed on 32-bit
- test: use integers of different sizes, even on 32-bit
- test: make 32-bit version of failed test

- ssa: relax what can be promoted
- doc: is_int_type() returns false for SYM_RESTRICTs
- enum: add testcase for UB in oversized shift
- enum: fix UB when rshifting by full width
- enum: add testcase for type of enum members
- enum: add testcase for base & enumerator type
- enum: fix cast_enum_list()
- enum: use the smallest type that fit
- enum: use the values to determine the base type
- enum: only warn (once) when mixing bitwiseness
- enum: warn when mixing different restricted types
- enum: warn on bad enums
- enum: rewrite bound checking
- enum: keep enumerators as int if they fit
- enum: default to unsigned
- enum: more specific error message for empty enum
- __attribute__((fallthrough)) can't simply be ignored
- ptrlist: add ptr_list_nth_entry()
- add testcase for missing function designator expansion
- fix expansion of function designator
- teach sparse about '-o <file>'
- teach sparse about '-x <language>'
- cgcc: add support to ignore argument(s) of options
- cgcc: teach about '-o <file>'
- cgcc: teach about '-x c'
- dump-macro: break the loop at TOKEN_UNTAINT
- dump-macro: simplify processing of whitespace
- fix implicit K&R argument types
- Use -Wimplicit-int when warning about missing K&R argument types
- Accept comma-separated list for function declarations.
- cgcc: use 'i386' for the arch instead of 'i86'
- add testcase for missing deliminator ' or "
- man: add section about reporting bugs
- man: add AUTHORS section
- man: update maintainer info
- don't allow newlines inside string literals
- multi-buffer for idents
- as-name: add and use show_as()
- as-name: use idents for address spaces

- as-name: allow ident as address_space
- as-name: check for multiple address spaces at parsing time
- as-named: warn on bad address space
- add detection of native platform
- Consolidate 'machine detection' into "machine.h"
- test endianness with __BYTE_ORDER__
- testsuite: test predef macros on LP32/LP64/LLP64
- fix '__SIZE_TYPE__' for LLP64
- allow optional "_T" suffix to __SIZEOF_XXX__
- use bits_mask() for predefined_max()
- add builtin_type_suffix()
- teach sparse about asm inline
- remove duplicates from gcc-attr-list.h
- show-parse: strip do_show_type()'s trailing space
- make predefined_type_size() more generic
- give a type to wchar
- use the type for predefined_max()
- add predefined macros for wint_t
- add predefined macros for [u]intptr
- add predefined macros for [u]intmax
- add predefined macros for [u]int{8,16}_t
- add predefined macros for [u]int64_t
- add predefined macros for [u]int32_t
- add predefined macros for char{16,32}_t
- fix the size of long double
- add predefine for __CHAR_UNSIGNED__
- add predefine_min() and use it for __{WCHAR,WINT}_MIN__
- add a flag to warn on casts to/from bitwise pointers
- show-parse: don't display null ident in show_typename()
- show-parse: do not display base type's redundant specifiers
- show-parse: remove string_ctype from typenames
- VERSION=0.6.0-rc1
- build: only need includedir from llvm-config
- build: check if sparse-llvm needs libc++
- remove unneeded declarations in "compat.h"
- remove unused arg in add_branch()
- allocate BBs after the guards
- remove redundant check of _Bool bitsize
- remove unused regno()

- remove -finline-functions from CFLAGS
- remove self-assignment of base_type
- doc: fix list formatting
- as-name: document that identifiers are OK for address spaces
- add TODO list.
- Sparse v0.6.0

Ramsay Jones (9):

- Makefile: use locally built sparse in the selfcheck target
- sparsec: use a compatible exception model on cygwin
- sparsei: add the –[no-]jit options
- constant: add -Wconstant-suffix warning
- pre-process: suppress trailing space when dumping macros
- pre-process: print macros containing # and ## correctly
- pre-process: don't put spaces in macro parameter list
- pre-process: print variable argument macros correctly
- pre-process: add the -dM option to dump macro definitions

Randy Dunlap (6):

- sparse: minor manpage corrections
- Documentation: make data-structures.txt easier to read
- Documentation: editing fixes in test-suite
- lib.c: early return from handle_onoff_switch()
- sparse: ignore indirect_branch attribute
- sparse: option to print compound global data symbol info

Thiebaud Weksteen (1):

- Add testcases for bitwise cast on pointer

Tycho Andersen (1):

- expression.h: update comment to include other cast types

Uwe Kleine-König (6):

- build: make PREFIX overwritable from the environment
- build: put comment about local.mk to the place where it is included
- build: drop BASIC_CFLAGS and ALL_CFLAGS
- build: drop -g from LDFLAGS
- build: pass CPPFLAGS to compiler
- build: only generate version.h when needed

Vincenzo Frascino (1):

- print address space number for cast-from-AS warnings

8.1.4 v0.5.2 (2018-04-30)

The latest release of sparse have been pushed to the official repository. It's a smaller release than the previous one but it contains some important changes to not be flooded by unimportant warnings while compiling the kernel.

The most notable changes are:

- better tracking and handling of expression constness
- fix bug with variadic macros
- less warnings on unknown attributes (none by default now)
- teach sparse about __builtin_{isinf_sign,isfinite,isnan}
- various update to the documentation
- do selfcheck with the locally built sparse
- some fixes or improvements for build (armhf, GNU/kfreebsd, . . .)
- also evaluate files included via -include

Many thanks to everyone involved.

Luc Van Oostenryck

—

Al Viro (1):

- Sparse preprocessing bug with zero-arg variadic macros

Christopher Li (8):

- gcc attr: add nonstring warn_if_not_aligned
- Makefile: provide CFLAGS for command line override.
- Give the constant pseudo value a size
- sparse-llvm: use pseudo->size to select llvm integer type
- Update gcc attribute list
- Fix crash cause by previous pseudo size change

Jacob Keller (1):

- sparse: document that -Wbitwise is default

Logan Gunthorpe (1):

- add __builtin functions for isinf_sign, isfinite and isnan

Luc Van Oostenryck (13):

- constexpr: rename handle_simple_initializer() to handle_initializer()
- constexpr: collect storage modifiers of initializers
- return an error if too few args
- give default return type in evaluate_call()
- constexpr: flag __builtin_bswap() as constexpr
- build: disable sparse-llvm on non-x86
- fix cgcc ELF version for ppc64/pcc64le
- fix: missing evaluate with '-include' : add testcase
- fix: missing evaluate with '-include'
- Revert "Give the constant pseudo value a size"

- By default disable the warning flag '-Wunknown-attribute'
- bump up version to 0.5.2-RC1
- Sparse v0.5.2

Martin Kepplinger (2):

- compile-i386.c: fix a memory leak in sort_array()
- compile-i386: make use of expression_list_size()

Nicolai Stange (20):

- constexpr: introduce additional expression constness tracking flags
- constexpr: init flags at expression allocation
- constexpr: examine constness of casts at evaluation only
- constexpr: examine constness of binops and alike at evaluation only
- constexpr: examine constness of preops at evaluation only
- constexpr: examine constness of conditionals at evaluation only
- constexpr: add support for tagging arithmetic constant expressions
- constexpr: add support for tagging address constants
- constexpr: check static storage duration objects' intializers' constness
- constexpr: recognize static objects as address constants
- constexpr: recognize address constants created through casts
- constexpr: recognize address constants created through pointer arithmetic
- constexpr: recognize members of static compound objects as address constants
- constexpr: recognize string literals as address constants
- constexpr: recognize references to labels as address constants
- constexpr: examine constness of __builtin_offsetof at evaluation only
- constexpr: flag builtins constant_p, safe_p and warning as constexprs
- constexpr: relax some constant expression rules for pointer expressions
- constexpr: support compound literals as address constants
- constexpr: treat comparisons between types as integer constexpr

Ramsay Jones (1):

- Makefile: use locally built sparse in the selfcheck target

Randy Dunlap (5):

- sparse: minor manpage corrections
- Documentation: make data-structures.txt easier to read
- Documentation: editing fixes in test-suite
- test-suite: handle format with filename.c not existing
- sparse: ignore indirect_branch attribute

Uwe Kleine-König (4):

- build: remove version.h in clean target
- cgcc: teach cgcc about GNU/kFreeBSD
- compile-i386: Use SPARSE_VERSION instead of __DATE__

- cgcc: provide __ARM_PCS_VFP for armhf

8.1.5 v0.5.1 (2017-08-18)

It is finally there. Sparse 0.5.1 is released.

I consider this the best quality of release of sparse I ever experienced so far. There are lots of enhancement and bug fixes incorporate into this release.

I would like to thank every one that contributes to this release, people who submit patches, perform testing and send bug reports.

I want to specially thank Luc Van Oostenryck who makes this release possible. He has 242 commits in this release, far more than any one else.

The development effort for the next release has already began.

Finally, I would like to call for developers joining the sparse project. If you are interested in modern compilers, reading compiler books and want some excise. Sparse is a good project to start.

Sparse is very small, the project compiles under 10 seconds. It can digest the full kernel source file, generate internal byte code representation and perform check on it. All this happens in 1/10 of the time it took gcc to compile the same source file.

Here is some project ideas, https://git.kernel.org/pub/scm/devel/sparse/sparse.git/tree/Documentation/project-ideas.md?h=v0.5.1

Thanks

Chris

Aaro Koskinen (1):

- build: allow use of PKG_CONFIG to override pkg-config

Andy Shevchenko (1):

- lib.c: skip –param parameters

Ard Biesheuvel (2):

- sparse: treat function pointers as pointers to const data
- Ignore pure attribute in assignement

Azat Khuzhin (2):

- sparse, llvm: compile: skip function prototypes to avoid SIGSEGV
- validation/prototype: regression for skipping prototypes

Christian Borntraeger (1):

- s390x: add the proper defines for data types

Christopher Li (24):

- Minor clean up for option handling
- round up the array element size to byte align
- Make same_symbol list share the same scope
- rename -Werror to -Wsparse-error
- teach next_designators() use array_element_offset()
- Ptr list sorting should use memmove instead of memcpy
- Make macro expanded string immutable
- Fix warning compiling sparse-llvm

- Adding ignored attribute optimize
- Let create_symbol check for previous same symbol
- Add full list of gcc attribute
- bump sparse's version to 0.5.1-rc4
- Adding gcc attribute no_gccisr
- Add test case for the wine dead loop bug
- Makefile: clean up and simplify
- Makefile: add selfcheck target
- Adding _Pragma()
- fix warnings report by selfcheck
- Adding gcc attribute noipa etc
- Adding document for sparse patch submit process
- Documents: project ideas
- test-inspect: handle special case iter==NULL
- test-inspect: Detect gtk3 then gtk2 package
- Sparse 0.5.1

Cody P Schafer (3):

- build: allow use of LLVM_CONFIG to override llvm-config config script
- sparse{i,c}: use LLVM_CONFIG to find llc and lli
- parse: support c99 [static ...] in abstract array declarators

Dan Carpenter (1):

- ptrlist: reading deleted items in NEXT_PTR_LIST()

Daniel Wagner (1):

- parse: Add comment to struct statement

Edward Cree (1):

- Allow casting to a restricted type if !restricted_value

Emilio G. Cota (1):

- Define __CHAR_BIT__

Emily Maier (2):

- linearize: Emit C99 declarations correctly
- validation: Check C99 for loop variables

Hans Verkuil (3):

- Add test case for extern array
- Add test case for anonymous union initializer
- Add test case for the ioc type check

Heiko Carstens (1):

- sparse/parse.c: ignore hotpatch attribute

Jeff Layton (2):

- sparse: make bits_to_bytes round up instead of down

- Handle SForced in storage_modifiers

Joe Perches (1):

- sparse: Allow override of sizeof(bool) warning

Johannes Berg (1):

- implement constant-folding in __builtin_bswap*()

John Keeping (3):

- validation/sizeof-bool: fix broken test case
- evaluate: split out implementation of compatible_assignment_types
- Support GCC's transparent unions

Lance Richardson (3):

- sparse: ignore __assume_aligned__ attribute
- sparse: update __builtin_object_size() prototype
- sparse: add support for _Static_assert

Linus Torvalds (5):

- Add warning about duplicate initializers
- Use any previous initializer to size a symbol
- Fix error at anoymous unions
- Fix scoping of extern symbols in block scope
- Fix initializers in anonymous structs and unions

Luc Van Oostenryck (242):

- Teach sparse about the __COUNTER__ predefined macro
- Fix size calculation of unsized bool array
- Do not drop 'nocast' modifier when taking the address.
- fix mixup in "Handle SForced in storage_modifiers"
- Fix type checking of variadic functions
- add missing #include "char.h" to char.c
- make 'ignored_attributes[]' static
- cleanup: remove evaluate_arguments()'s unused argument
- Warn on unknown attributes instead of throwing errors
- Remove unneeded variable in integer_promotion()
- fix discarded label statement
- add test case for builtin bswap with constant args
- make ptrlist walking against robust against empty blocks
- let "compile" not crash on bools
- give comparable label's names to basic blocks
- OP_SWITCH should use 'insn->cond' instead of 'insn->target'
- remove unused field 'multijump' in struct instruction
- storage should not be inherited by pointers
- testsuite: simplify test function-pointer-inheritance

- use a shorter name for function-pointer-modifier-inheritance.c
- testsuite: test modifiers preserved by '&' operator
- testsuite: test modifiers preserved by 'typeof()'
- some modifiers need to be preserved by 'typeof()'
- Update maintainers in the manpage
- cgcc should not define non-reserved identifiers
- recursive phi_defines cannot happen
- fix missing element in types declaration
- add support for __int128
- fix typing error in compound assignment
- llvm: fix typing when comparing to a constant
- llvm: remove unneeded OP_COPY support
- fix cast to bool
- unssa: do not try to update liveness
- unssa: simplify rewrite of OP_PHISOURCE
- unssa: try to avoid some OP_PHI copies
- unssa: eliminate trivial phisrc copies
- unssa: update comment about the unneeded copies
- volatile loads must not be simplified
- fix superfluous phisrc
- fix phisrc mixup
- missing load simplification
- fix value of label statement
- C11: teach sparse about '_Thread_local'
- C11: teach sparse about '_Noreturn'
- C11: teach sparse about '_Alignof()'
- C11: teach sparse about '_Alignas()'
- C11: teach sparse about '–std={c11,gnu11}'
- fix cast's target type info
- fix crash while testing between conditional & unconditional OP_BR
- kill uses of replaced instructions
- fix killing OP_PHI instructions
- fix killing OP_CAST & friends
- fix killing OP_SELECT
- fix killing OP_COMPUTEDGOTO
- explicitely ignore killing OP_ENTRY
- cleanup kill_instruction()
- fix conditional context test case with void
- add helper: is_scalar_type()

- validate expression's type in conditionals
- remove unused arg in uses/defs functions
- add testcase for wrong early escape conversion
- warn on unknown escapes after preprocessing
- remove 'Escape' from token character class
- fix killing OP_SETVAL instructions
- define __LP64__ & _LP64 if arch_m64 is enabled
- add an helper for common predefined macros
- define __LONG_MAX__ & __SIZEOF_POINTER__
- move OP_MUL simplification in a separate function
- simplify '(x / 1)' to 'x'
- simplify '(x * -1)' to '-x'
- simplify '(x / -1)' to '-x' (but only for signed division)
- simplify '(x % 1)' into '0'
- simplify '~(~x)' and '-(-x)' to 'x'
- simplify '(x || 1)' to '1'
- simplify '(x op x)' to '0', '1' or 'x'
- add warning option '-Wtautological-compare'
- simplify comparisons followed by an equality test against 0 or 1
- simplify '(x || x)' and '(x && x)'
- add support for LLP64 arch
- move evaluation & expansion of builtins in a separate file
- let identical symbols share their evaluate/expand methods
- expand __builtin_bswap*() with constant args
- testsuite: give a proper name to the 'binary-constant' test
- testsuite: make tests known to fail effectively fail
- testsuite: simplify the ioc-typecheck case
- testsuite: add a simple test for -Wenum-mismatch
- testsuite: add tag to ignore the output/error
- testsuite: report as error tests known to fail but which succeed
- allow to launch the test suite from the project root dir
- testsuite: check patterns presence or absence in output
- testsuite: add some selfchecking
- testsuite: check the nbr of times a pattern should be present
- testsuite: use 'error' instead of 'info' for successful tests known to fail
- testsuite: get 'check-known-to-fail' earlier
- testsuite: allow quieter error reporting
- testsuite: quieter error reporting for 'known-to-fail'
- cleanup: there is no 'struct phi' to allocate

- remove unused field ‘multijmp’ in struct statement
- remove unused field ‘goto_bb’ in struct statement
- fix show-parse()’s labels
- add killing of OP_SLICEs
- add killing of OP_PHISOURCEs
- add helper kill_use_list()
- fix killing of OP_PHIs
- fix clear_phi(), replace it by kill_instruction()
- remove unused clear_phi()
- fix killing of otherwise not-handled instructions
- kill_instruction() may need to be forced or not
- add killing of pure calls
- fix killing OP_CALL via pointers
- add killing of non-volatile loads
- add killing of stores
- fix killing of rewritten loads
- use kill_instruction() when killing an OP_PHI during CSE
- use kill_instruction() when killing any instructions during CSE
- fix OP_PHI usage in try_to_simplify_bb()
- simplify float-to-float casts that doesn’t change size
- CSE: add test cases for comparisons duality
- CSE: use commutativity to identify equivalent instructions
- CSE: avoid hashing removed instructions
- fix expansion cost of pure functions
- add missing braces around FOR_EACH_PTR loop
- make -Wbitwise operational again
- use option: ‘-Woverride-init’
- add test case for warnings about overlapping initializers
- allow to warn on all overlapping initializers
- fix checking of overlapping initializer
- ignore whole-range overlapping initializer
- fix usage in simplify_seteq_setne()
- fix size of loaded bitfields
- split OP_BR between unconditional & conditional: OP_CBR
- remove unused helper is_branch_goto()
- replace test for c99 for-loop initializers
- add test case for scope of C99 for-loop declarations
- add test cases for storage of c99 for-loop declarations
- add an optional validation method to external_declaration()

- check the storage of C99 for-loop initializers
- move 'extern with initializer' validation after the validate method
- use VOID instead of directly using &void_pseudo
- teach sparse about -Waddress
- add is_func_type()
- warn if testing the address of a function
- add is_array_type()
- warn if testing the address of an array
- fix evaluation of a function or array symbol in conditionals
- fix is_scalar_type()
- fix test for cast to bool on 32bit machines
- predefine __INT_MAX__ and friends
- predefine __SIZEOF_INT__ & friends
- fix test validation/div.c
- fix cast to pointer to floating-point
- do not depends on limits.h to test __CHAR_BIT__
- fix expansion of integers to floats
- avoid crash with test-linearize -vv
- fix OP_PHI usage in try_to_simplify_bb(), correctly
- be more careful with concat_user_list()
- avoid useless warning for 'bool <- restricted type' conversion
- introduce REPEAT_CFG_CLEANUP
- let kill_unreachable_bbs() clear REPEAT_CFG_CLEANUP
- fix: kill unreachable BBs after killing a child
- ignore VOID when trying to if-convert phi-nodes
- fix boolean context for OP_AND_BOOL & OP_OR_BOOL
- fix missing reload
- keyword: add test case for reserved '_Static_assert'
- keyword: regroup the [reserved] keywords
- keyword: explicitly add C99 & C11 keywords
- keyword: add more reserved keywords to the test case
- keyword: add a comment about NS_TYPEDEF & reserved keywords
- keyword: no pre-declaration needed for attribute names
- add get_<allocator>_stats()
- add show_allocation_stats()
- add helper handle_simple_switch()
- teach sparse how to handle '-fmem-report'
- use -fmem-report to report allocation stats
- testsuite: cleanup result files

- fix: kill old branch in insert_branch()
- returns the correct type when evaluating NULL
- remove bit_size & bit_offset from struct access_data
- add test case for linearize_initializer() of bitfields
- fix implicit zero initializer.
- remove alignment from struct access_data
- remove origval from struct access_data
- add support for a new flag: -fdump-linearize[=only]
- more tests for implicit 'bool <- restricted' casts
- avoid warning on explicit 'bool <- restricted' casts
- define ident_list
- teach sparse how to dump macro definitions
- fix hardcoded size of wide chars
- avoid to redefine __INT_MAX__ and friends
- fix definition of __SCHAR_MAX__ & friends
- teach sparse how to handle -dD flag
- let -dD report macro definitions
- testsuite: get all tags in once
- testsuite: grep the expected output only when needed
- testsuite: grep the output patterns only when needed
- testsuite: use shell arithmetic instead of fork-execing expr
- testsuite: remove unneeded './' before commands
- testsuite: avoid fork+execing basename
- teach cgcc about OSX aka darwin
- ret-void: add test case for toplevel asm
- ret-void: warn for implicit type
- use NULL instead of 0 in testcases.
- finer control over error vs. warnings
- Add more declarations for more builtin functions
- keep the warnings table alphabetically sorted
- cgcc: alphasort warning names in check_only_option()
- cgcc: add missing warning names to check_only_option()
- cgcc: filter-out '-fdump-linearize[=...]'
- memcpy()'s byte count is unsigned
- add support for -Wmemcpy-max-count
- add support for -fmemcpy-max-count
- fix: add missing examine in evaluate_dereference()
- fix OP_PHI usage in try_to_simplify_bb() only when non-bogus
- fix: try_to_simplify_bb eargerness

- add fallback for missing __builtin_bswapXX()
- fix: __builtin_bswap{16,32,64}() constantness
- dissect: use built_in_ident() instead of MK_IDENT()
- teach sparse about -m{big,little}-endian
- teach sparse about __{BIG,LITTLE}_ENDIAN__
- teach sparse about __BYTE_ORDER__ & __ORDER_{BIG,LITTLE}_ENDIAN__
- cgcc: teach cgcc about arm64
- cgcc: teach cgcc about ppc64[le]
- cgcc: teach cgcc about arm
- bump sparse's version to -rc3
- fix ptrlist corruption while killing unreachable BBs
- fix infinite simplification loops
- fix BB dependencies on phi-nodes
- fix crash when ep->active is NULL
- fix crash in rewrite_branch()
- fix some crashes in add_dominators()
- fix crash with sym->bb_target == NULL
- take comma expr in account for constant value
- fix: give a type to bad cond expr with known condition
- ptrlist: add a counter for the number of removed elemnets
- ptrlist: adjust ptr_list_size for the new ->rm field
- ptrlist: add MARK_CURRENT_DELETED
- ptrlist: avoid iteration on NULL entries
- mark pseudo users as deleted instead of removing them
- testsuite: add support for commands with timeout
- Remove single-store shortcut
- Bump sparse's version to -rc5
- Sparse v0.5.1

Michael Stefaniuc (3):

- Add the __builtin functions needed for INFINITY and nan().
- Add a define for __builtin_ms_va_copy()
- Add tests for the builtin INF and nan() functions.

Oleg Nesterov (3):

- dissect: teach do_expression() to handle EXPR_OFFSETOF
- dissect: teach do_initializer() to handle the nested EXPR_IDENTIFIER's
- dissect: s/mode_t/usage_t/ in report_member()

Omar Sandoval (1):

- sparse-llvm: Fix LLVM 3.5 linker errors

Pavel Roskin (1):

- Use LLVM_CONFIG instead of llvm-config in Makefile

Ramsay Jones (15):

- Add the __restrict__ keyword
- sparse: add 'gnu_inline' to the ignored attributes
- don't call isdigit/tolower with a char argument
- Makefile: suppress error message from shell
- don't run sparse{c,i} tests when sparse-llvm is disabled
- Add support for multiarch system header files
- cgcc: use only the cc command to determine $gcc_base_dir
- cgcc: use $ccom to set $multiarch_dir if not specified
- test-suite: remove bashism to avoid test failures
- cgcc: avoid passing a sparse-only option to cc
- parse.c: remove duplicate 'may_alias' ignored_attributes
- compile-i386.c: don't ignore return value of write(2)
- sparse: add 'alloc_align' to the ignored attributes
- lib: workaround the 'redeclared with different type' errors
- Makefile: pass -Wno-vla to sparse while checking pre-process.c

Randy Dunlap (1):

- documentation: update email reference link

Rui Teng (1):

- sparse: add no_sanitize_address as an ignored attribute

Thomas Graf (1):

- sparse: Make -Werror turn warnigns into errors

Tony Camuso (2):

- .gitignore: add cscope and Qt project files
- Add default case to switches on enum variables

8.1.6 v0.5.0 (2014-04-01)

As a lot of you have already noticed. The sparse 0.5.0 has been tagged and upload to the kernel.org for a while.

One of the most noticeable change of the sparse 0.5.0 is the license. The sparse project has finally move to the MIT license. Thanks for the hard work of Dan Carpenter and Franz Schrober, who contact and collect permissions from the sparse developers. We actually manage to get *all* developer's blessing on the MIT license.

This release also has some enhancement matching the latest kernel source annotations. It will reduce the noise level of the sparse warnings.

So there you have it. The official announcement of the sparse 0.5.0.

Chris —

Al Viro (9):

- Fix handling of __func__
- Fix tab handling in nextchar_slow()
- Fix ,##__VA_ARGS__ kludge

- Gentler handling of bitwise warnings in unary operations
- simplify handling of newline/whitespace flags in expand()
- fix handling of -include
- massage handling of wide string literals/character constants in tokenizer
- switch to delayed handling of escape sequences
- L ## 'a' is valid; so's L ## "a"

Benjamin Herrenschmidt (1):

- sparse, llvm: Fix varargs functions

Christopher Li (19):

- Limit usage of g++ to llvm related programs.
- Merge branch 'sparse-llvm' of git://github.com/penberg/sparse-llvm.git
- Adding default for m64/m32 handle
- Merge branch 'for-chris' of git://github.com/penberg/sparse-llvm
- Merge branch 'llvm/core' of github.com:penberg/sparse-llvm
- remove weak define x86_64
- Merge git://git.kernel.org/pub/scm/linux/kernel/git/viro/sparse into marge
- Clean up some test case error.
- Fix segfault cause by fucntion without ident.
- Get rid of gcc warning about enum values
- Larger buffer size for token concatenation
- Proper variable length array warning
- Allow forced attribute in function argument
- Trivial: Remove redundant Makefile variable
- Merge branch 'llvmcore'
- Merge branch 'Novafora' of git://git.zytor.com/users/hpa/sparse/sparse into license
- Sparse 0.5.0 rc1
- Fix make dist failure
- Sparse 0.5.0

Emilio G. Cota (1):

- gitignore: add 'version.h'

Ethan Jackson (1):

- sparse: Add 'leaf' to ignored attributes.

Franz Schrober (5):

- Revert "Update the information in README about using the library."
- Revert "Fix mistaken comparison that becomes a no-op."
- sparse: Relicense under the MIT license
- FAQ: Remove outdated sections about the license
- sparse: Also check bit_offset when checking implicit casts

Frederic Crozat (1):

- Add __builtin_stpcpy, __sync_synchronize, __sync_bool_compare_and_swap to declare_builtin_functions

James Westby (1):

- Update the information in README about using the library.

Jan Pokorný (2):

- unssa: track use of newly added pseudo
- simplify: conservative handling of casts with pointers

Jeff Garzik (15):

- sparse, llvm: if-else code generation
- sparse-llvm: OP_SEL
- sparse-llvm: OP_SWITCH
- sparse-llvm: OP_LOAD
- sparse-llvm OP_PHISOURCE: replace copy with target=src pointer operation
- sparse, llvm: replace FIXME comment with assert(), following

existing style

- sparse, llvm: implement OP_CALL
- sparse, llvm: move OP_PHI code from switch statement to separate function
- sparse, llvm: move OP_CAST code to separate func. support FP casts.
- sparse, llvm: create helper for obtaining instruction's type
- sparse, llvm: store module-local functions on function reference list
- sparse, llvm: move OP_COPY support to separate function. Add FP support.
- sparse, llvm: support OP_STORE
- sparse, llvm: Fix loops, by properly handling OP_PHI forward references
- sparse, llvm: add loop testcase

Jeff Layton (1):

- sparse: add __builtin_va_arg_pack() and __builtin_va_arg_pack_len()

Joe Perches (1):

- There's no current way to know the version of sparse. Add –version to see it.

Jonathan Neuschäfer (9):

- FAQ: update the website address and call it Wiki
- ptrlist.c: fix a typo in a comment
- sparse, llvm: 'Verify' the LLVM module before writing it
- sparse, llvm: convert the condition of branch/select to bool
- sparse, llvm: Fix type of loaded values
- sparse, llvm: Fix resulting type of store address calculations
- sparse, llvm: de-duplicate load/store address calculation code
- sparse, llvm: base load/store address type on insn_symbol_type()
- sparse, llvm: add a struct access test case

Josh Triplett (2):

- Define __SIZEOF_POINTER__
- Support #pragma once

KOSAKI Motohiro (2):

- sparse: Add '__vector_size__' to ignored attributes
- sparse: Add 'error' to ignored attributes

Kamil Dudka (2):

- cse: treat PHI-nodes as other instructions
- cse: update PHI users when throwing away an instruction

Kim Phillips (2):

- sparse: add built-in byte swap identifiers
- sparse: add built-in atomic memory access identifiers

Linus Torvalds (1):

- sparse, llvm: Make function declaration accessible to backend

Masatake YAMATO (3):

- Warn about initialization of a char array with a too long

constant C string.

- Test case for -Winit-cstring option
- Add description for -Winit-cstring option

Mauro Dreissig (1):

- Fix wrong array size expression

Pekka Enberg (69):

- sparse, llvm: Initial commit
- sparse, llvm: Fix assert() in sparse code
- sparse, llvm: Fix global variable initialization
- sparse, llvm: Fix 'sparsec' when it's not in PATH
- llvm, sparse: Separate entry and exit basic blocks
- sparse, llvm: Add switch statement to output_insn()
- sparse, llvm: OP_RET/PSEUDO_VAL code generation
- sparse, llvm: Add support for OP_RET/PSEUDO_ARG
- sparse, llvm: Introduce 'struct function' to clean up code
- sparse, llvm: Add output_op_binary() stub
- sparse, llvm: Implement OP_ADD
- sparse, llvm: Add support for more binary ops
- sparse, llvm: Implement some binary comparison ops
- sparse, llvm: Move binop tests to validation/backend
- sparse, llvm: Implement OP_CAST
- sparse, llvm: Floating point support for binops
- sparse, llvm: Reorganize code generation tests
- sparse, llvm: Bitwise not operator codegen

- sparse, llvm: Kill ifdef'd unssa() call
- sparse, llvm: Kill debugging code
- Merge pull request #1 from jgarzik/hacks
- Merge pull request #2 from jgarzik/hacks
- sparse, llvm: Warn the user when we fall back to GCC
- sparse, llvm: Code generation for string constants
- sparse, llvm: Cleanup output_data()
- sparse, llvm: Fix OP_CAST to use zero-extend
- sparse, llvm: Improve sparsec front-end
- sparse, llvm: Fix PSEUDO_OP code generation
- sparse, llvm: Don't redefine module local functions
- Revert "sparse, llvm: Don't redefine module local functions"
- sparse, llvm: Fix code generation for casts
- sparse, llvm: Fix pseudo_type() for PSEUDO_ARG
- Merge pull request #3 from jgarzik/hacks
- Merge branch 'master' of github.com:penberg/sparse-llvm
- llvm, sparse: Fix symbol_is_fp_type() goof
- Merge pull request #4 from jgarzik/hacks
- sparse, llvm: Fix code generation for 'long double' data type
- sparse, llvm: Add support for struct types
- sparse, llvm: Add support for symbol initializers
- sparse: Bump up sizeof(_Bool) to 8 bits
- sparse, llvm: Add support for logical ops
- sparse, llvm: Fix 'void *' pointer code generation
- sparse, llvm: Use new LLVM type system API for structs
- sparse, llvm: Fix struct code generation
- sparse, llvm: Fix symbol_type() for bitfields and enums
- sparse, llvm: Add support for array types
- sparse, llvm: Add support for union types
- sparse, llvm: Make 'sparsec' error handling more robust
- sparse, llvm: Function pointer code generation
- sparse, llvm: Fix symbol initializer code generation
- sparse, llvm: Fix 'extern' symbol code generation
- sparse, llvm: Make llc output to stdout in sparsec
- sparse, llvm: Pointer cast code generation
- sparse, llvm: OP_SET_B and OP_SET_A code generation
- sparse, llvm: More comparison ops code generation
- sparse, llvm: Simplify comparison op code generation
- sparse, llvm: FP comparison op code generation

- Merge pull request #6 from jgarzik/hacks
- sparse, llvm: Don't fail the build if LLVM is too old
- sparse, llvm: Use LLVMInt1Type() in sym_basetype_type()
- sparse, llvm: Add test case for <stdbool.h> type
- Revert "sparse: Bump up sizeof(_Bool) to 8 bits"
- sparse, llvm: Add _Bool to cast validation test
- sparse, llvm: Simplify output_data() type logic
- sparse, llvm: Fix string initializer code generation
- sparse, llvm: Fix global string access code generation
- sparse, llvm: Fix SIGSEGV for extern symbols
- sparse, llvm: Fix 'void' return type code generation
- sparse, llvm: Use LLVM_HOSTTRIPLE

Ramsay Jones (2):

- char.c: Fix parsing of escapes
- symbol.c: Set correct size of array from parenthesized string initializer

Randy Dunlap (1):

- sparse patch v2: add noclone as an ignored attribute

Robert Bedichek (1):

- Novafora license grant using MIT license.

Shakthi Kannan (1):

- I have updated the sparse.1 man page including the __bitwise relevant content, and created Documentation/sparse.txt with the complete comparison between __nocast vs __bitwise.

Xi Wang (17):

- compile-i386: fix use-after-free in func_cleanup()
- check missing or duplicate goto labels
- fix casting constant to _Bool
- fix SIGFPE caused by signed division overflow
- sparse, llvm: fix link errors
- sparse, llvm: fix phi generation
- sparse, llvm: simplify function generation
- sparse, llvm: improve pointer arithmetic handling
- sparse, llvm: set target specification
- sparse, llvm: use LLVM_DEFAULT_TARGET_TRIPLE
- sparse, llvm: fix array size
- sparse, llvm: cache symbol_type() result
- sparse, llvm: fix struct name generation
- sparse, llvm: set more data attributes
- sparse, llvm: die if error
- Fix result type of relational and logical operators

- Fix expression type for floating point negation ('!')

8.1.7 v0.4.5-rc1 (2013-05-09)

I just push the tag for sparse 0.4.5-rc1

It is about time to cut a new release. There have been a lot of small improvements. It produces less warning on the recent kernel check.

Please give it a good test.

Chris

8.1.8 v0.4.4 (2011-11-25)

This is a long due release. The sparse 0.4.4 can be downloaded from:

https://www.kernel.org/pub/software/devel/sparse/dist/

The new sparse has a lot of bugs fixes. It will report less noise while checking the new kernel tree. It compiles better with the new gcc as well.

The sparse project is in the process of moving to the MIT license. Dan is coordinating the efforts. Most sparse developers sign off the MIT license already. If you haven't done so, please contact Dan off the list regarding the new license.

Here is the short summery of the changes in this release.

Have a nice long week end.

Chris

Ben Pfaff (1):

- evaluate: Allow sizeof(_Bool) to succeed.

Christopher Li (13):

- inspect: adding function arugument list
- Allow overwrite CFLAGS from command line
- Ignore attribute vector_size
- Remove set but not used variable
- inspect: Add switch statement and more
- validation: inline switch statement
- Fix inlining switch statement.
- Sparse 0.4.4-rc1
- Add test case for empty asm clobbers
- Fix parsing empty asm clobber
- Sparse 0.4.4-rc2
- Add test case for binary constants
- sparse 0.4.4

Dan Carpenter (1):

- recognize binary constants

Diego Elio Pettenò (3):

- build: allow easy override of GCC_BASE

- build: add an all-installable target that builds the targets to install.
- Fix build with GCC 4.6 series.

Florian Fainelli (1):

- Makefile: warn user when libxml and/or libgtk2 are not available

Jan Pokorný (4):

- remove unused "container" macro
- flow.c: make comment for 'dominates' reflect code
- use ARRAY_SIZE() when possible (continued)
- parse.c: "if(" -> "if (" adjustment

Jonathan Neuschäfer (3):

- fix a memory leak in compile-i386.c
- FAQ: fix a typo ("because or")
- fix common misspellings with codespell

Kamil Dudka (2):

- cse: treat PHI-nodes as other instructions
- cse: update PHI users when throwing away an instruction

Linus Torvalds (5):

- Add new streams to a hash-list based on their names
- Teach 'already_tokenized()' to use the stream name hash table
- Make 'linearize_iterator()' helper function
- Make 'linearize_switch()' helper function
- Make 'linearize_return()' helper function

Michael Stefaniuc (1):

- Ignore the ms_hook_prologue attribute.

Namhyung Kim (3):

- use ARRAY_SIZE() when possible
- Fix tokenizer for octal escape sequences
- Update the validation check for escape sequences

Nicolas Kaiser (1):

- memops.c: always true expression

Pekka Enberg (4):

- sparse: Add 'artifical' to ignore attributes
- sparse: Enable unhandled validation tests
- Show expected vs. actual output on test failure
- sparse: Fix __builtin_safe_p for pure and const functions

8.1.9 v0.4.3 (2010-11-02)

Hi,

It is final there. The sparse version 0.4.3 is released.

Mostly small fix up. It can parse the recent kernel better, less noise.

For people interested in the sparse internals, there is a sparse inspecting tools now. Currently it has limited knowledge of AST. It is very easy to extent though.

Thanks every one for the contribution.

Chris

–

Bernd Petrovitsch (1):

- Fix a typo - "typdef" is neither C nor plain English

Christopher (3):

- evaluate: check for NULL type inside typeof
- Add test case for builtin_unreachable()
- inspect: add some expression inspection

Christopher Li (15):

- Make MOD_NORETURN fits into 32 bit
- Move noreturn attribute out of ignore attr area
- Declare ignored attributres into a list of string.
- Simplify Makefile using static pattern rules
- Adding test case for "x && y && z" .
- Pointer don't inherent the alignment from base type
- Allow parsing L'0'
- Parsing wide char string
- Adding asm goto label test case
- inspect: add custom ast treeview model
- inspect: add some example inspect for symbol and statement
- inspect: Add test-inspect program
- inspect: cast expression
- Fixup and cleanup modifier_string() function.
- sparse 0.4.3 finial

Damien Lespiau (1):

- Ignore the may_alias GCC attribute

Dan Carpenter (1):

- add test-inspect to .gitignore

Dan McGee (1):

- Makefile: fix permissions mixup on install

Daniel De Graaf (1):

- Fix incorrect linearization of "x && y && z"

Jiri Slaby (3):

- parser: add support for asm goto
- parser: fix and simplify support of asm goto
- parser: define __builtin_unreachable

Joel Soete (1):

- possible fix to cgcc issue in sparse 0.4.2:

Josh Triplett (2):

- Rename -Wall to Wsparse-all, so it doesn't get turned on unintentionally
- New attribute designated_init: mark a struct as requiring designated init

Kamil Dudka (1):

- do not ignore attribute 'noreturn'...

Michael Buesch (2):

- ignore attributes "externally_visible" and "signal"
- Ignore "naked" attribute

Michael Stefaniuc (3):

- Ignore the ms_abi/sysv_abi attributes.
- Ignore the alloc_size attribute.
- Handle __builtin_ms_va_list.

Mike Frysinger (1):

- parser: add Blackfin gcc info

Morten Welinder (1):

- skip may_alias and declare builtin_fabs

8.1.10 v0.4.2 (2009-10-16)

I have tagged and released the sparse version 0.4.2 at http://ftp.be.debian.org/pub/software/devel/sparse/dist/ with sha1sum a2adef3f78c7409e8c0bb80941f473d775afcac4

As previously discussed on the sparse mailing list, I am the new maintainer of the sparse project. This is my first release for sparse. Thanks Josh Triplett for the previous maintenance of the project.

I also created a new sparse wiki, it will replace the current sparse home page. https://sparse.wiki.kernel.org/

A lot of bug fixes and enhancements have gone into this release. Special thanks to Al Viro for overhauling the parser. Now sparse has better ctype and attribute handling. The detailed changes follow.

Al Viro (39):

- saner warnings for restricted types
- fix show_typename()
- catch !x & y brainos
- fun with declarations and definitions
- Fix type_info_expression()
- fun with declarations and definitions
- Fix handling of ident-less declarations
- Separate parsing of identifier-list (in K&R-style declarations)

- More nested declarator fixes
- Fix attribute/asm handling
- more direct_declarator() sanitizing
- Warn about non-empty identifier list outside of definition
- Apply attributes after (to the right place
- Leave applying attributes until we know whether it's a nested declarator
- Don't mess with passing symbol to declarator/direct_declarator
- Fix braino in which_kind()
- Sanitize direct_declarator logics
- Separating ctype and parser state, part 1
- Propagate decl_state to declaration_specifiers()
- Fix regression created by commit af30c6df74f01db10fa78ac0cbdb5c3c40b5c73f
- Take the rest of storage class keywords to parse.c
- Fix handling of typedefs with several declarators
- preparations to ->declarator() cleanup - separate typedef handling
- Take the rest of specifiers to parse.c
- Saner type for __builtin_va_list
- Rewrite and fix specifiers handling
- Have ->declarator() act directly on ctype being affected
- Clean up and split declaration_specifiers()
- Pass decl_state down to ->declarator() and handle_attributes()
- Pass decl_state down to ->attribute()
- Restore __attribute__((mode)) handling
- Fix enumeration constants' scope beginning
- Fix declaration_specifiers() handling of typedef name shadowed by NS_SYMBOL
- Fix __label__ handling
- Simplify get_number_value() and ctype_integer()
- Don't mix storage class bits with ctype->modifiers while parsing type
- Sanitize pointer()
- Segfault at evaluate.c:341
- warn directive in argument list

Alberto Bertogli (1):

- Support the __thread storage class

Alexander Shishkin (1):

- don't call sparse when called to generate dependencies

Alexey Zaytsev (16):

- Remove symbol.id_list
- Replace the -specs cgcc option with -target
- Make show_symbol newline-consistent

- Handle a terminal -o option properly.
- Looks more evident this way.
- Mark handle_switch as static and don't export it from lib.h
- Handle missing argument to -D.
- Gdb macros to get a better look at some sparse data structures.
- A slightly edited irc discussion with Josh Triplett.
- Warning should be enough for an unhandled transparent union
- Set gcc include path at runtime.
- Let cgcc pass -gcc-base-dir to sparse.
- Document -gcc-base-dir in sparse.1
- Rename dirafter to idirafter.
- Let void have sizeof 1
- Also warn about sizeof(function)

Blue Swirl (6):

- Sparc64 (Sparc V9, LP64) support
- OpenBSD support
- Ignore attribute __bounded__, used by OpenBSD headers.
- Add c{l,t}z{,l,ll}, ffsl{,l}, popcountll and floating point comparison builtins.
- Add support for TImode type (__int128_t)
- Define __LP64__ for x86_64 unless in 32 bit mode

Christopher Li (11):

- Evaluate iterator symbols
- Remove pre_buffer
- Add enum member list to the parent
- Teach classify_type to handle typeof
- Warn about explicit usage of sizeof(void)
- Makefile automatic header dependency
- Clean up Makefile long lines
- Update the validation check for ftabstop=
- Add validation for restrict and attribute warning
- move extern inline function to file scope
- Sparse 0.4.2

David Given (2):

- Unhardcode byte size being 8 bits.
- Add type information to struct instruction.

Geoff Johnstone (4):

- Add support for GCC's -std=... and -ansi command line options.
- Add builtin functions for use with __FORTIFY_SOURCE
- Fix type mismatches with incomplete types

- Add -Wno-declaration-after-statement

Hannes Eder (4):

- Add -ftabstop=WIDTH
- refactor handle_switch_f
- test-suite: be more verbose on 'unhandled' and 'known to fail' tests
- test-suite: integrate unhandled proprocessor tests

Johannes Berg (8):

- cgcc: handle ppc arch
- make sparse keep its promise about context tracking
- sparse test suite: add test mixing __context__ and __attribute__((context(...)))
- sparse: simple conditional context tracking
- inlined call bugfix & test
- improve -Wcontext code and messages
- fix bug in context tracking code
- Revert the context tracking code

Josh Triplett (2):

- Add test case for new warning about !x & y
- Expand "dubious !x & y" handling to other combinations of !, &, and |.

Kamil Dudka (4):

- compile-i386: do not generate an infinite loop
- linearize.h: sanitize header
- unssa: track uses when replacing a phi node
- make sparse headers self-compilable...

Linus Torvalds (5):

- Fix cast instruction generation
- Simplify (and warn about) right shifts that result in zero
- Allow array declarators to have 'restrict' in them
- Turn off '-Wtransparent-union' by default
- Avoid "attribute 'warning': unknown attribute" warning

Martin Nagy (3):

- .gitignore: Ignore dependencies and Vim swap files
- Add missing checks for Waddress-space
- Print an error if typeof() lacks an argument

Pavel Roskin (1):

- Ignore "cold" and "hot" attributes, which appeared in gcc 4.3

Pekka Enberg (1):

- sparse: Add GCC pre-defined macros for user-space

Ramsay Jones (1):

- Makefile: suppress error message from pkg-config

Reinhard Tartler (1):

- show_token: handle TOKEN_UNTAINT and TOKEN_ARG_COUNT types

Samuel Bronson (1):

- Have Makefile import local.mk if it exists.

Thomas Schmid (1):

- Fix implicit cast to float

Vegard Nossum (2):

- Fix use of invalid file descriptor
- Set *tree to NULL on error

– Chris Li

8.1.11 v0.4.1 (2007-11-13)

I have tagged and tarballed Sparse 0.4.1, now available from http://ftp.be.debian.org/pub/software/devel/sparse/dist/ with sha1sum 14085c5317cd7f2c8392fb762969906fa91888ef.

This bugfix release fixes a Sparse assertion which recent Linux kernels started triggering, along with a few other fixes.

Full changelog:

Christopher Li (1):

- Perform local label lookup

Emil Medve (1):

- Handle ignored attribute malloc

Josh Triplett (4):

- Add comment on taint flags enum referencing expr->taint
- Add test-suite metadata to validation/local-label.c
- Add known-to-fail test case for a static forward declaration
- Makefile: VERSION=0.4.1

Mike Frysinger (1):

- fix install perms of manpages

Tilman Sauerbeck (1):

- Added a prototype for mempcpy().

– Josh Triplett

8.1.12 v0.4 (2007-09-15)

I have tagged and tarballed Sparse 0.4, now available from http://kernel.org/pub/software/devel/sparse/dist/sparse-0.4.tar.gz, with sha1sum a77a10174c8cdb5314eb5000c1e4f24458848b91.

Highlights and visible changes in this release:

- The Sparse validation files have become an automated test suite, invoked via 'make check'. Thanks to Damien Lespiau for the initial 'test-suite' script. Also thanks to Pavel Roskin for ideas, discussion, and prototyping, and to the contributors of new test cases and test-suite metadata for existing test cases.

- New backend 'c2xml' by Rob Taylor, which outputs an XML representation of the variable declarations, function declarations, and data structures in a C file.
- 'sparse' and 'cgcc' now have manpages.
- **Warning changes:**
  - Warn on 'return <void expression>;' if given '-Wreturn-void'; off by default because the C99 standard permits this.
  - Sparse now defaults to -Wno-do-while.
  - Sparse now defaults to -Wdecl.
  - -Wno-old-initializer turns off warnings about non-C99 struct initializers
  - -Wno-non-pointer-null turns off warnings about using a plain integer as a NULL pointer
  - Initializer entries defined twice and bitfields without explicit signs have changed from errors to warnings.
- 'cgcc' no longer passes '-Wall' to 'sparse' if given '-Wall' on the command line. '-Wall' turns on all Sparse warnings, including experimental and noisy ones. Don't include it just because a project wants to pass '-Wall' to 'cc'. If you really want 'cgcc' to run 'sparse' with '-Wall', set 'CHECK='sparse -Wall''
- Support for more builtin functions: '__builtin_labs', '__builtin_offsetof', '__builtin_strcat', '__builtin_strncat', and '__builtin_strlen'.
- Support for more attributes: 'constructor', 'destructor', '__always_inline__', '__noinline__', 'used', '__syscall_linkage__', 'format_arg', 'cdecl', 'stdcall', 'fastcall', 'dllimport', and 'dllexport'.
- Support for the '__DATE__' and '__TIME__' preprocessor symbols.
- Support for '__alignof' (treated like the existing '__alignof__').
- typeof(bitwise_type) now produces the same type, not an incompatible type; thanks to Al Viro.
- Various profile-driven optimizations and changes to compiler optimization flags, making Sparse significantly faster.
- **Numerous fixes to C standards compliance, thanks to Al Viro. In particular:**
  - Sparse now requires integer constant expressions in various places, not arbitrary expressions.
  - Sparse now handles NULL pointer constants more correctly.
  - Sparse now resolves pointer types more correctly in conditional expressions
- The 'graph' backend now outputs significantly better program graphs, and the Sparse source code includes some 'gvpr' scripts to modify these graphs in various ways, such as only showing the callers or callees of particular functions. Thanks to Dan Sheridan.
- Linux-style 'make' output; for a more verbose make with full command lines, 'make V=1'. Thanks to Damien Lespiau.
- Numerous bugfixes and internal cleanups; thanks to the many Sparse contributors.

Full changelog:

Al Viro (51):

- handle __alignof as equivalent of __alignof__
- saner reporting of overlaps in initializers
- check for whitespace before object-like macro body
- fix alignment for _Bool
- fix interaction of typeof with bitwise types
- better recovery from bad operations on bitwise
- make copying of EXPR_INDEX non-lazy

- tie the fields of struct in simple list
- rewrite of initializer handling
- fix handling of typeof on structs
- missing NULL checks in initializer handling
- take cast_to() out of usual_conversions(), do it in callers
- mechanically split compatible_assignment_types()
- null pointer constants have no special meaning for pointer subtraction
- remove long-dead variable in evaluate_ptr_add()
- remove useless argument in evaluate_ptr_sub()
- cleanup of evaluate_assign_op()
- clean up the typechecking in arithmetics
- clean up usual_conversions(), kill evaluate_shift()
- fix index conversions in evaluate_ptr_add()
- fix default argument promotion
- move degenerate() down into compatible_assignment_types()
- in case of compound literal we want to delay examining type
- warn on return <void expression>;
- deal with enum members without excessive PITA
- implement __builtin_offsetof()
- fix handling of integer constant expressions
- fix the comma handling in integer constant expressions
- first pass at null pointer constants
- make size_t better approximate the reality
- fix handling of address_space in casts and assignments
- fix handling of pointers in ?:
- saner show_type()
- clean up evaluate_sign()
- integer_promotions() can't get SYM_NODE or SYM_ENUM
- start cleaning type_difference()
- get compatible_assignment_types() deal with all cases
- fix the sanity check in evaluate_ptr_sub()
- rewrite type_difference()
- deal correctly with qualifiers on arrays
- add __builtin_strlen()
- no such thing as array of functions
- new helper: unfoul()
- handling of typeof in evaluate_member_dereference()
- file and global scopes are the same for purposes of struct redefining
- . . . ,array should degenerate

- sanitize evaluate_ptr_add(), start checking for pointers to functions
- fix evaluate_compare()
- sanitize evaluate_postop()
- saner -Wtypesign
- braino in conditional_expression()

Alberto Bertogli (1):

- Implement x86-64 support in cgcc.

Alexey Dobriyan (2):

- Fix infinite loop in free_preprocessor_line()
- Fix -E handling

Christopher Li (2):

- combinations string clean up
- Pass a bitmask of keywords to handle_attributes

Damien Lespiau (6):

- Change sparse homepage in ctags headers.
- __DATE__ & __TIME expansion
- Beautify all & install Makefile targets
- test-suite: a tiny test automation script
- test-suite documentation
- Sample test-suite test cases

Dan Sheridan (2):

- Improved graph generation using subgraph clusters for functions
- Add gvpr-based post-processing for graphs

Josh Triplett (118):

- Fix website and repository references in FAQ
- Fix the version number
- Remove old version note.
- gitweb lives at git.kernel.org now.
- Add a "make dist" that requires $(VERSION) to match 'git describe'
- Add test case for __asm__ __volatile__(...)
- Make cgcc not pass -Wall to sparse even if passing it to cc
- Teach cgcc about all currently existing sparse warning options
- Teach cgcc about -ventry and -vdead
- Parse asm after a label as a statement, not an attribute
- Add test case for stdcall and cdecl attributes.
- Add -Wno-old-initializer to turn off warnings about non-C99 struct initializers
- Add test case for -Wno-old-initializer
- Revert unintentional inclusion of warning fix in previous commit.
- Use %td when printing a ptrdiff_t to avoid problems on 64-bit platforms

- Remove extra space.
- Add shebang to gvpr scripts, make them executable, and change usage accordingly
- Fix an __attribute__() parsing error
- Expand calling convention test case to cover fastcall
- Add -Wno-non-pointer-null to turn off warning about using a plain integer as a NULL pointer
- Add __builtin_strcat and __builtin_strncat.
- Ignore the GCC constructor and destructor attributes
- Remove inaccurate comment designating some attributes as windows-specific.
- Move the ident for defined() into the preprocessor section.
- Reorganize attribute list for readability.
- Add double-underscore variant __always_inline__.
- Add double-underscore variant __noinline__.
- Add no-double-underscore variant "used", ignored like "__used__".
- Add double-underscore variant __syscall_linkage__.
- Add no-double-underscore variant format_arg.
- Add explanatory comment about direct use of __IDENT for preprocessor idents.
- Sparse always defines __STDC__ 1, so cgcc does not need to do so
- Fix old typo: s/wierd/weird/
- Canonicalize URL in FAQ: add www., add trailing slash
- Change "LD" to "LINK" in Makefile prettyprinting.
- Makefile prettyprinting: make INSTALL and other output line up correctly
- Add test case for infinite loop in free_preprocessor_line()
- Turn on -Wdecl by default.
- ctags: Use const as appropriate in cmp_sym()
- validation/old-initializer.c: Make the_s static to avoid extraneous warning.
- validation/restricted-typeof.c: Make globals static to avoid extraneous warnings.
- validation/escapes.c: Make globals static to avoid extraneous warnings.
- validation/non-pointer-null.c: Make global static to avoid extraneous warning.
- Merge commit 'viro/integer-constant'
- Move all the preprocessor tests into validation/preprocessor/
- Move test-suite output files to validation/.gitignore
- .gitignore: Stop ignoring all dotfiles
- validation: Update comments for current Sparse behavior and test-suite.
- Add test-suite comments to all the obvious preprocessor tests
- Make preprocessor-loop a normal numbered preprocessor test
- Add test-suite comment to preprocessor21.
- Add test-suite comment to address_space.c
- Make clean depend on clean-check
- Rename asm-volatile to better describe what it tests

- Add test-suite comment to label-asm.c
- Remove “check-exit-value: 0” and rely on default; remove extra blank line.
- Add test-suite comment to bad-array-designated-initializer.c
- Add c2xml to .gitignore
- Split c2xml build rule into compile and link stages, and add the quiet prefixes
- expression.h needs lib.h for struct position and symbol.h for int_ctype
- Fix GCC warnings in c2xml
- Fix sparse warnings in c2xml: mark globals static and remove unused globals
- Fix test-suite to handle stdout and stderr separately, and fix up tests
- Add test-suite metadata to bad-cast.c
- Add test-suite metadata to bad-ternary-cond.c, and remove now-redundant comment
- Add test-suite metadata to initializer-entry-defined-twice.c
- Add test-suite metadata to context.c
- Add test-suite metadata to escapes.c
- Add test-suite metadata to calling-convention-attributes.c
- Fix typos in test-suite documentation
- Makefile: stop cleaning files we didn’t make and have no business cleaning
- Add test-suite metadata to old-initializer.c; also test with -Wno-initializer
- allocate.h: Stop needlessly returning a void value in __DO_ALLOCATOR
- Turn off -Wdo-while by default.
- Add test-suite metadata to label-attr.c
- validation/builtin_safe1.c: Show the unsafe macro argument
- Make “Initializer entry defined twice” a warning, not an error
- Remove explicit restatements of defaults in metadata for member_of_typeof test
- Remove explicit restatements of defaults in metadata for outer-scope test
- Remove explicit restatements of defaults in metadata for comma test
- Add test case for comparing null pointer constant to int.
- Makefile: Use -O2 -finline-functions, not just -O
- cse: Size insn_hash_table more realistically, speeding up CSE significantly
- Add some missing dependencies in the Makefile
- Drop -fpic; it hurts performance and we don’t build libsparse.so by default
- Add another test case to validation/comma.c
- ctags: Handle some new namespaces and symbol types.
- is_zero_constant: declare saved const
- Add test case for -Wtypesign
- Sort warning options in lib.c and lib.h
- Rename Wcast_to_address_space to Wcast_to_as to match the command-line argument
- Add a manpage for sparse
- Install the Sparse manpage

- cgcc: Sparse accepts -Wcast-to-as, not -Wcast-to-address-space
- Rename Wundefined_preprocessor to Wundef to match the command-line argument
- cgcc: Sparse accepts -Wundef, not -Wundefined-preprocessor
- Use -fno-strict-aliasing, as the ptrlist code seems to violate C99 strict aliasing rules
- Add test-suite annotations to restricted-typeof.c
- Add test-suite annotations to double-semicolon.c
- Add test-suite annotations to check_byte_count-ice.c
- Add test-suite annotations to badtype4.c
- Add test-suite annotations to varargs1.c
- Add test-suite annotations to struct-attribute-placement.c
- Add test-suite annotations to non-pointer-null.c
- Add test-suite annotations to struct-ns1.c
- Add test-suite annotations to noderef.c
- Makefile: Use ?= to allow overriding OS or AR on the Make command line
- FAQ: Point to URL on vger for subscription instructions and archives
- README: recode from ISO-8859-1 to UTF-8
- validation: Rename typeconvert.c to integer-promotions.c to match its purpose
- Add test-suite annotations to integer-promotions.c
- Add test-suite annotations to cond_expr.c
- Add test-suite annotations to function-pointer-modifier-inheritance.c
- validation: Update comment in type1.c to reflect current state of Sparse
- Add test-suite annotations to init-char-array.c
- Add a manpage for cgcc
- Add SEE ALSO for cgcc in sparse manpage
- Makefile: VERSION=0.4

Kovarththanan Rajaratnam (1):

- libxml compile fix on Cygwin

Michael Stefaniuc (3):

- Ignore the cdecl and stdcall attributes for now.
- Add test for typedef on pointer to function with stdcall attribute.
- '?' is a valid escape character defined by ANSI C. Its value is '?'.

Mike Frysinger (1):

- Makefile: improve flag handling

Pavel Roskin (5):

- Improve error message if using a member of an incomplete struct or union
- Bitfield without explicit sign should be a warning, not an error
- cgcc: preserve sparse exit code if -no-compile is used
- Avoid use of libc headers in the validation suite
- Fix warnings about undeclared globals, they are irrelevant to the test

Ramsay Jones (2):

- Add (more) support for WIN32 attribute names
- Add cygwin support to cgcc

Randy Dunlap (1):

- add __builtin_labs()

Rob Taylor (4):

- add end position to symbols
- add sparse_keep_tokens api to lib.h
- new get_type_name function
- add c2xml program

Yura Pakhuchiy (1):

- Make cgcc filter out all sparse warning related options

ricknu-0@student.ltu.se (4):

- tokenize.c: Replace handwritten strncmp with existing function.
- expression.c: Clean up match_oplist() and add missing va_end()
- parse.c: Adding va_end().
- tokenize.c: Simplify drop_stream_eoln().

– Josh Triplett

8.1.13 v0.3 (2007-05-01)

I have tagged and tarballed a 0.3 release of Sparse, now available from http://ftp.be.debian.org/pub/software/devel/sparse/dist/ with sha1sum 1d868b29234176abd5f3f5463aad1f11d5268dc2.

Note that the Sparse Git repository has moved to:

https://git.kernel.org/cgit/devel/sparse/sparse.git

The old repository location will continue to work for now, but please update any references you have to the old location.

Thanks to Christopher Li for contributing heavily to this release, including several notable new features. See the full changelog for details.

In addition to numerous bug fixes, cleanups, and new test cases, this release includes several new visible features:

- A Sparse-based implementation of ctags, thanks to Christopher Li. Now you can have a ctags that 'actually parses C'.
- Sparse now correctly supports annotations on inline functions.
- The new '-ventry' option will make Sparse dump its internal linearized bytecode format.
- Sparse now parses the '__regparm__' attribute just like 'regparm', fixing a warning with pthreads from glibc 2.5.
- Sparse now interprets the integer size attributes 'QI', 'SI', 'HI', and 'DI', and their double-underscore variants.
- Sparse now supports attributes on labels, sometimes used to specify 'unused' labels.
- Sparse now handles structure attributes between the structure keyword and the name, fixing many parse errors.
- Sparse now supports more than one command-line include file.

- The pkg-config file gets installed to its correct location under 'lib', rather than under 'share'.

Notable internal changes:

- Fully table-driven attribute parsing. This simplifies the C parser and makes it far easier to add new attributes to Sparse.
- Many internal cleanups, more information preserved for backends, more useful formats for that information, and lower memory usage.
- Fixed Sparse warnings about itself.

Full changelog:

Christopher Li (24):

- Sparse-based Ctags implementation
- Change the symbol access list to a pseudo list
- Add instruction to pseudo user tracking.
- Update usage chain for dead instructions
- Update usage chain for dead branch instruction.
- Allow more than one command line include file.
- Enhance debug information.
- Another attempt to fix the attribute parsing.
- Marking anonymous string.
- Bug fix in pointer modifier inheritance at function degeneration.
- Handle structure attributes between the structure keyword and the name
- Fix the segfault when initializer has unknown symbol
- Fix double semicolon in struct declaration
- Make the ptrlist using the sparse allocator.
- Fix core dump on anonymous symbol.
- Fix a bug that match_idents forget to end with NULL
- Adding debug option for showing the linearized instruction.
- Fix core dump on huge switch
- Introduce expression_error
- Disable liveness "dead" instruction by default.
- Add annotation for inline function call.
- Introduce keyword driven attribute parsing
- Fix the annotated inline call position
- handle label attributes

Christopher Li and Josh Triplett (4):

- Introduce top level parsing for asm parsing.
- Introducing statement keywords
- Free up some special bits in modifiers.
- Moving statement parsing into smaller functions.

James Westby (2):

- Fix mistaken comparison that becomes a no-op.

- Update the information in README about using the library.

Josh Triplett (30):

- Add ctags to .gitignore
- Add a return in the last case of a switch; redundant but less error-prone.
- Coding style fix: in a pointer type, * goes with the name, not the type.
- Add missing #include "allocate.h" in linearize.h for DECLARE_ALLOCATOR.
- Add test case for function pointer modifier inheritance
- Add test case for structure attribute placement.
- Add test case for double semicolon in structure declaration.
- Coding style fix: use parentheses with sizeof
- Move pkg-config file to lib, rather than share
- Add static to declarations in test cases, to remove unrelated warnings.
- Fix typo in symbol.h: s/keywrods/keywords/
- Fix typos in comments
- Use GCC format and sentinel attributes on appropriate functions
- Fix two potential NULL pointer dereferences in dissect.c
- Avoid returning an uninitialized pointer from dup_list of an empty list
- Remove stray space from expand_compare in expand.c
- Prevent potential NULL pointer dereference in expand_compare
- Add test case for basic address_space annotations.
- Use noreturn on die() and error_die()
- Parse and ignore the __regparm__ attribute, just like regparm.
- Fix comment to reference #weak_define rather than #ifndef, matching code
- Teach cgcc about -Wtransparent-union and -Wno-transparent-union
- Declare die_if_error extern in lib.h
- Remove unused variable "include" from lib.c
- Declare do_error static
- Declare gcc_patchlevel extern in lib.h
- compile-i386.c: Declare regs_in_use static
- simplify.c: Declare delete_pseudo_user_list_entry static
- linearize: DECLARE_ALLOCATOR for asm_constraint and asm_rules
- Fix most -Wshadow warnings in Sparse.

Oleg Nesterov (3):

- dissect: cleanup report_implicit()
- dissect: fix multidimensional array initializer
- dissect: simplify lookup_member()

– Josh Triplett

8.1.14 v0.2 (2006-12-05)

I have tagged and tarballed a 0.2 release of Sparse, now available from http://ftp.be.debian.org/pub/software/devel/sparse/dist/, with sha1sum 1762fc609fe436e74b87356a52690b5f7bb40c81.

In addition to plenty of bug fixes, this release includes several notable new features:

- -Wall, thanks to Pavel Roskin
- '#strong_define' and '#strong_undef', thanks to Oleg Nesterov
- Argument parsing functions no longer mangle the argv passed to them, thanks to Christopher Li
- static library and header files now installed, along with a pkg-config file to find them
- Makefile now supports DESTDIR, useful for packagers

Full changelog:

Christopher Li (4):

- trivial fix for seg fault.
- Fix warning on self check.
- delay removing file scope
- cleanup write to argument array hack

Damien Lespiau (1):

- trivial: more .gitignore stuff

Josh Triplett (5):

- Update the FAQ: add sparse website and gitweb, update git URL, remove old BK url
- Rename "check.c" to "sparse.c" to match program name; update .gitignore
- Install static library and header files
- Generate and install a pkg-config file. Add DESTDIR support to Makefile.
- Remove old SCCS target from Makefile.

Nicolas Kaiser (1):

- double inclusions

Oleg Nesterov (7):

- use lookup_macro() in handle_undef()
- kill NS_INVISIBLEMACRO, introduce NS_UNDEF
- fix redefine of #weak_define
- fix 'weak' attribute loss
- prepare for #strong_{define,undef}
- implement #strong_define
- implement #strong_undef

Pavel Roskin (1):

- Support -Wall flag

– Josh Triplett

8.1.15 v0.1 (2006-11-06)

I have tagged and tarballed a 0.1 release of Sparse, now available from http://ftp.be.debian.org/pub/software/devel/sparse/dist/, with sha1sum 9e0a4d5abb8e8a4be4cf8d9fe632c69dbec3e242.

As discussed in [http://marc.info/?l=linux-sparse&m=116231992212971 Re: Official releases?], I've taken maintainership of sparse. Thanks to Linus Torvalds for his previous maintainership. As a result, this release comes from my sparse Git repository. You can find more information about obtaining sparse via Git at the [https://www.kernel.org/pub/linux/kernel/people/josh/sparse/ new sparse homepage].

In addition to all the work in the [https://www.kernel.org/pub/scm/devel/sparse/sparse.git/ previous Sparse repository], this release includes the following changes:

Adam DiCarlo (1):

- Add type information to enum mismatch warning

Al Viro (2):

- added a bunch of gcc builtins
- switch to hash-based get_one_special()

Josh Triplett (15):

- "Initializer entry defined twice" should not trigger with zero-size fields
- Fix incorrect symbol in comment on #endif for multiple-inclusion guard
- Add -Wno-uninitialized
- graph: Show position in basic block nodes
- bb_terminated: Use boundary values rather than specific opcodes
- Turn on -Wcontext by default
- Merge branch 'fix-defined-twice-error-on-empty-struct' into staging
- Merge branch 'graph' into staging
- merge branch 'more-warning-flags' into staging and fix conflicts
- merge branch 'no-semantic-h' into staging and fix conflicts
- Merge branch 'Wcontext-default' into staging
- Add test cases to validation/context.c for the Linux __cond_lock macro
- Merge branch 'context-test-cases-for-cond-lock' into josh
- Rename test case bad-assignement.c to bad-assignment.c, fixing the typo.
- Stop building and installing libsparse.so

Josh Triplett and Pavel Roskin (1):

- Recognize and ignore __alias__ and __visibility__

Pavel Roskin (4):

- Compile sparse executable under it's own name, not as "check"
- Add support for __builtin_strpbrk()
- Typo fixes
- Install cgcc on "make install", refactor installation code

Known issue with this release:

- Sparse does not produce the expected set of warnings for several of the validation programs, included in the sparse source in the directory 'validation/'. Some scripts should provoke warnings but don't, and others provoke warnings they shouldn't.

– Josh Triplett

CHAPTER 9

Indices and tables

- genindex

Index

Symbols

C

D

E

F

G

H

I

K

L

O

P

R

S

T

U

X