gcc link time optimization and the Linux kernel

Andi Kleen
Intel OTC

Apr 2013

andi@firstfloor.org

Acknowledgments

- Lots of people helped/contributed

- Ralf Baechle, Richard Biener, Tim Bird, Honza Hubicka, H.J. Lu, Joe Mario, Markus Trippelsdorf, Changlong Xie, others

Why LTO?

- Optimize over the whole binary
  - Not just function (3.0) or file (<4.5)
- Avoid inline dependency hell in header files
- Without changing Makefiles significantly

gcc 4.7+ LTO WHOPR crash course

- Compiler parses files, writes GIMPLE to object files (LGEN)
  - Function optimization summaries are computed
- Linker calls lto1 with all files for sequential whole program analysis (WPA)
  - Merges types, Generates global callgraph, IPA optimization summaries, writes partitions
- Generate code per partition in parallel (LTRANS)
  - Inline inside partition, run IPA optimizations for real, run per function optimizations, generate object code

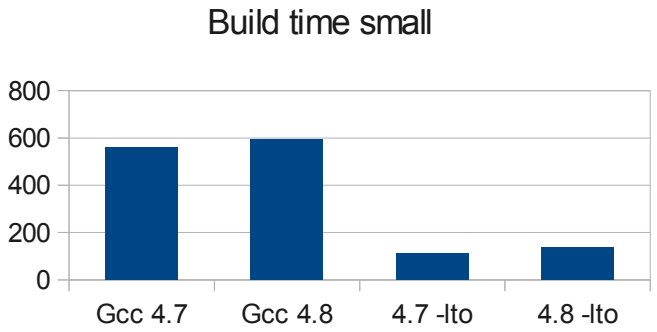
LTO IPA optimizations (4.8)

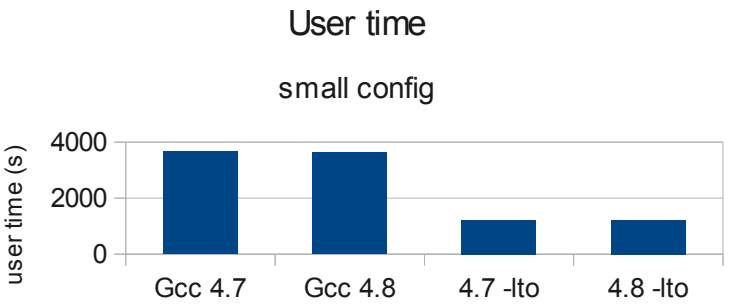
- Inlining between files
- Function cloning for specific arguments
- Remove unused code (fwhole-program)
- Increase alignment
- Discover pure/const
- Keep globals/statics alive over calls

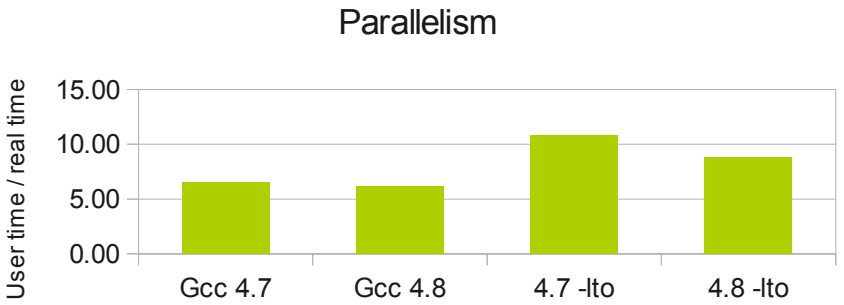
- De-virtualization
- Change ABI (SSE)
- Constant propagation
- Scalar replacement of aggregates
- Constructor / destructor merging

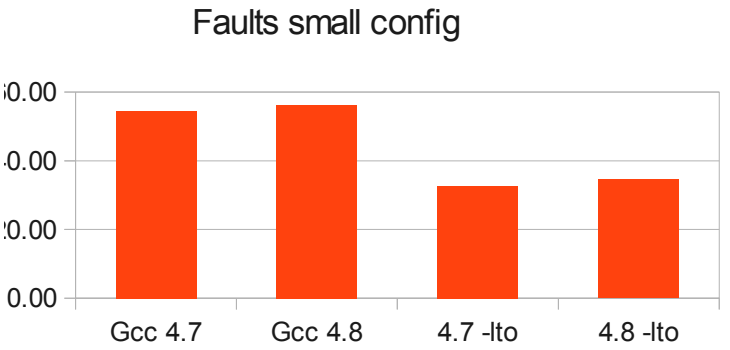
green: does not benefit kernel today

Build time





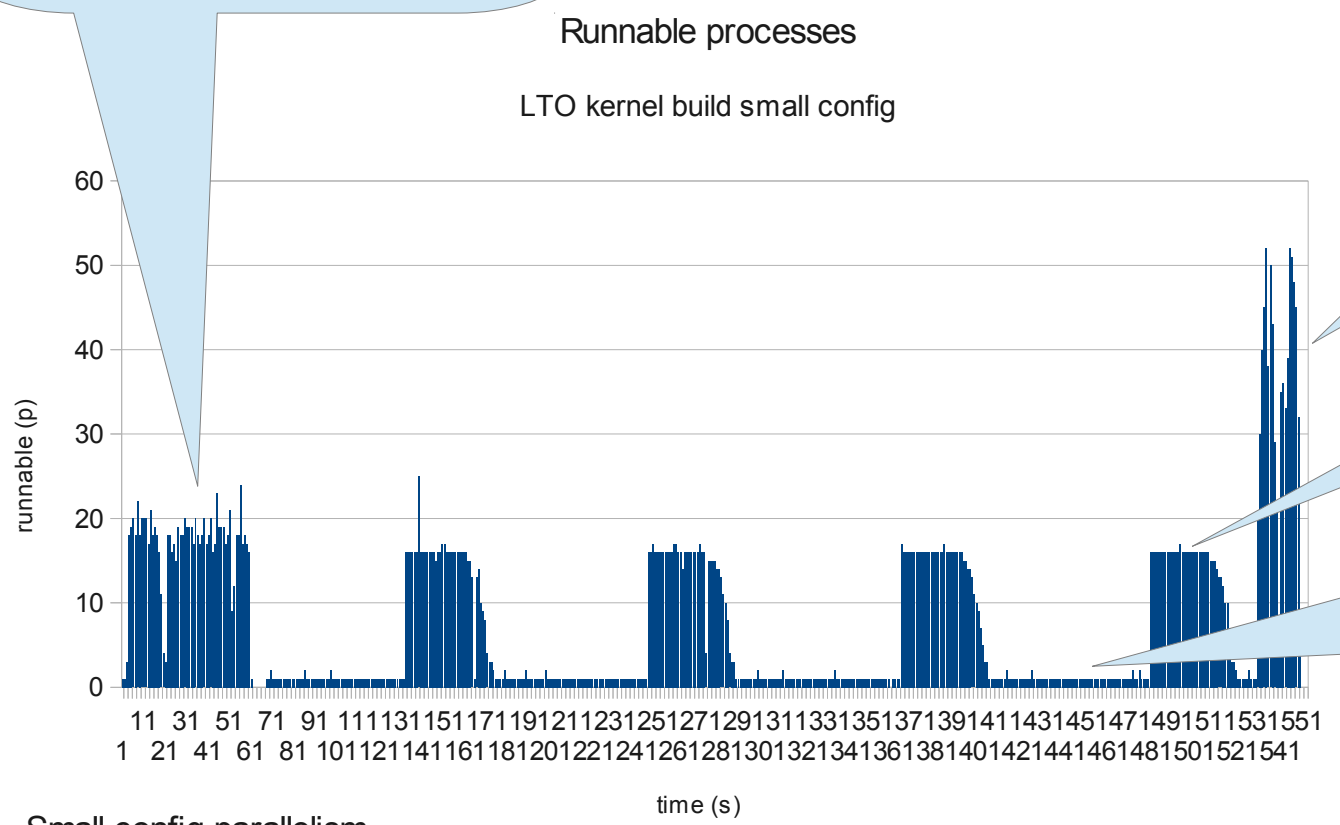




LTO is slow and not parallel enough

Parallelism small build 4.7

Object file generation
Parsing / LGEN

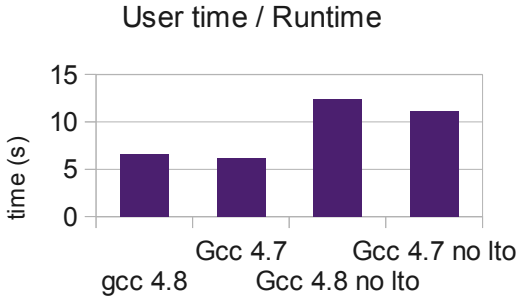


Modules without job server

LTRANS code generation

WPA + real linker
Type merging
4 times

Small config parallelism

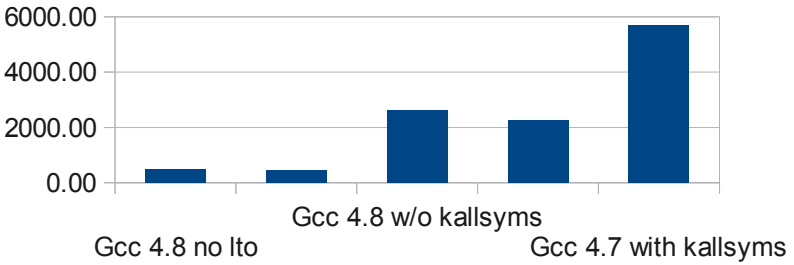


WHOPR still has poor parallelism

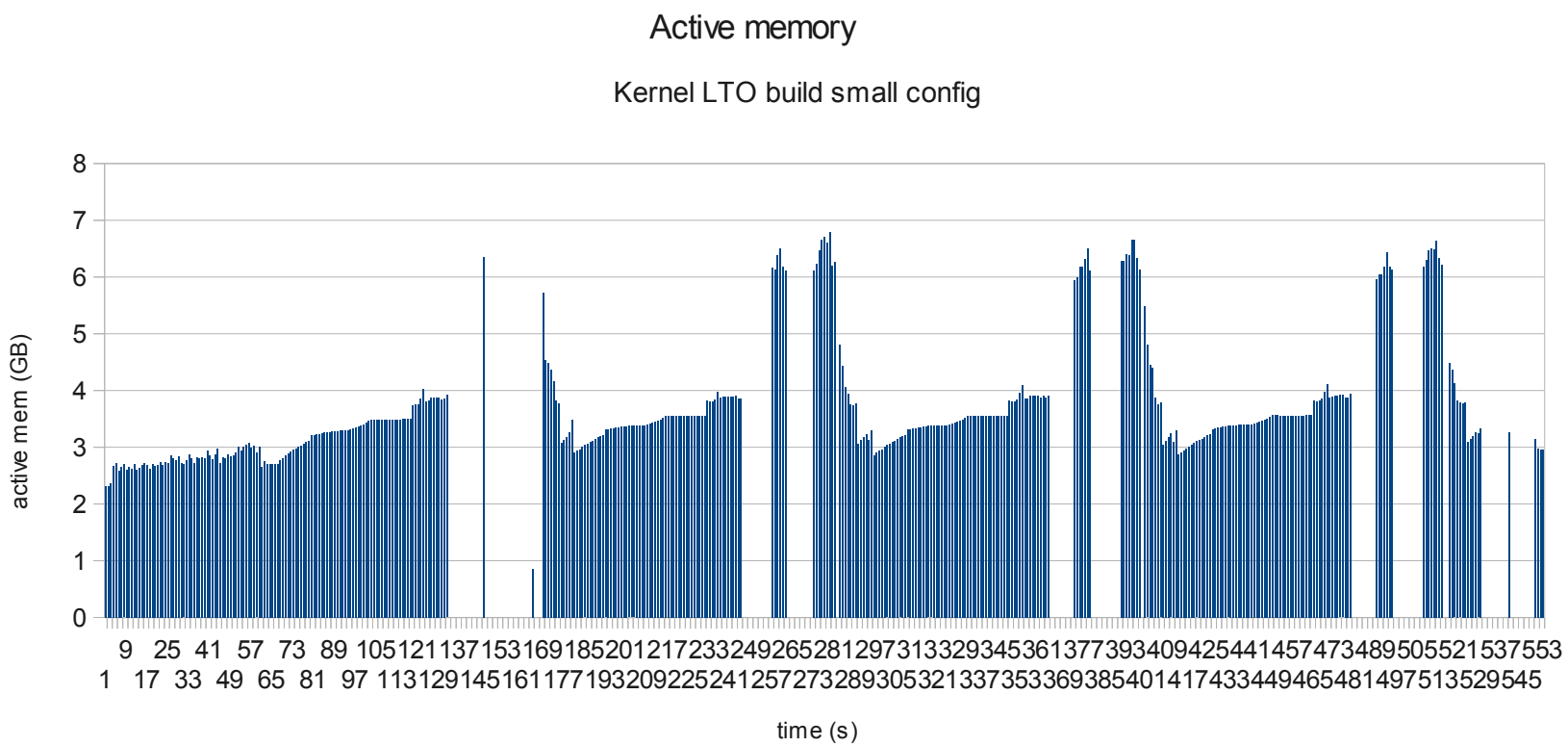
Multilink

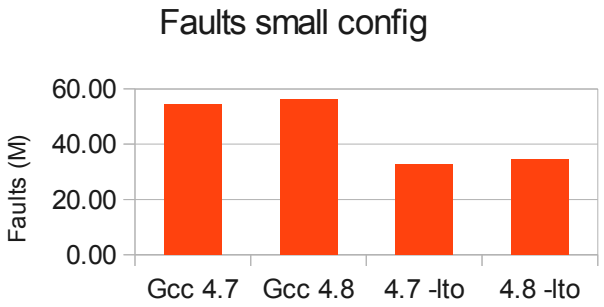
- vmlinux links 2-4x: runs LTO that often
- Generates integrated symbol table (kallsyms)
- So far not fixed
  - KALLSYMS can be disabled

- One of those unexpected quirks of real build systems



Memory usage 4.7 small build



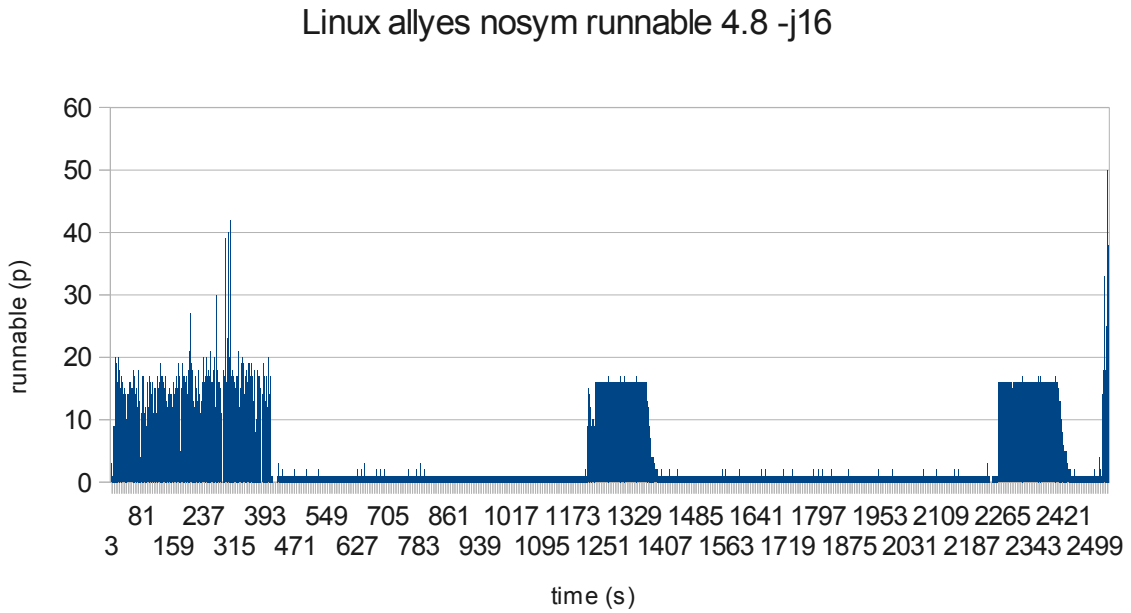


Even small build swaps in 4GB system
Memory peaks all in WPA

Memory consumption

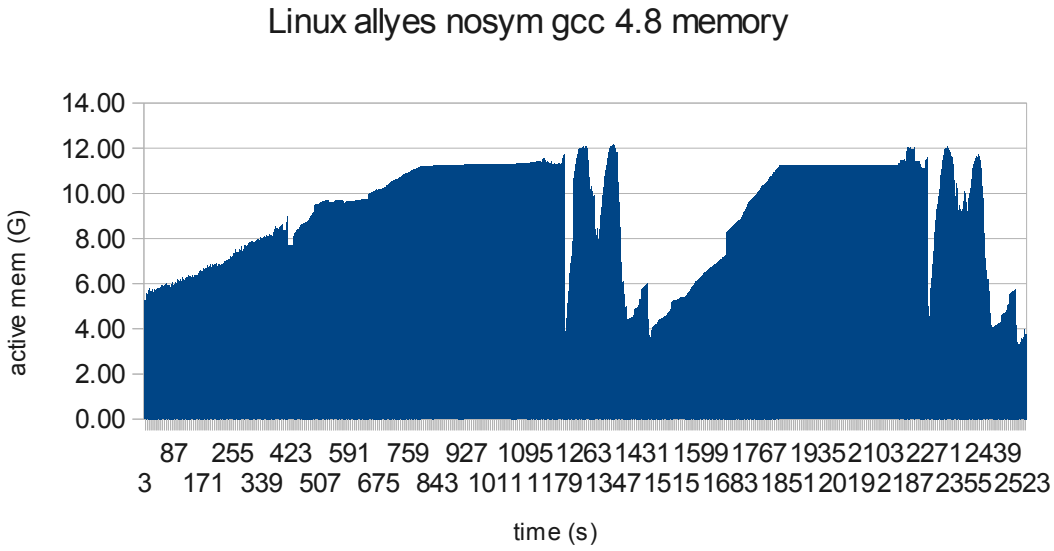
- Temporary data can be a problem, together with WPA
  - Early many swap storms with /tmp = tmpfs
  - Partitioning algorithm was improved
  - Use TMPDIR=objdir
  - With modules need to avoid too large -j* for parallel WPA
  - Jobserver has to be disabled, makes it worse

Large build 4.8 (allyes)



Kallsyms disabled
(disables various features)

~ 42mins
~1h 11min with kallsyms
~15GB peak



WPA cycle profile 4.8

```
  Samples: 257K of event 'cycles', Event count (approx.): 183142453583
- -   9.71%  lto1-wpa  lto1              [.] htab_find_slot_with_hash          ◆
-    - htab_find_slot_with_hash                                               ▒
-       + 61.79% gimple_type_hash(void const*)                               ▒
-       + 26.54% gimple_register_type_1(tree_node*, bool)                    ▒
-       + 4.79% visit(tree_node*, sccs*, unsigned int, vec<tree_node*, va_heap, v▒
-       + 1.53% iterative_hash_canonical_type(tree_node*, unsigned int)      ▒
-       + 0.98% iterative_hash_gimple_type(tree_node*, unsigned int, vec<tree_nod▒
-       + 0.92% symtab_get_node(tree_node const*)                            ▒
-       + 0.77% gimple_type_eq(void const*, void const*)                     ▒
-       + 0.53% build_int_cst_wide(tree_node*, unsigned long, long)          ▒
-       + 0.53% streamer_string_index(output_block*, char const*, unsigned int, b▒
- +   9.00%  lto1-wpa  libc-2.17.so       [.] _int_malloc                      ▒
- +   7.34%  lto1-wpa  lto1              [.] iterative_hash_hashval_t(unsigned i▒
- +   7.33%  lto1-wpa  lto1              [.] gimple_type_eq(void const*, void co▒
- +   6.66%  lto1-wpa  libc-2.17.so       [.] __memset_sse2                    ▒
- +   5.35%  lto1-wpa  libc-2.17.so       [.] malloc_consolidate               ▒
- +   4.11%  lto1-wpa  libc-2.17.so       [.] _int_free                        ▒
- +   4.10%  lto1-wpa  lto1              [.] tree_map_base_eq(void const*, void ▒
- +   2.85%  lto1-wpa  lto1              [.] pointer_map_insert(pointer_map_t*, ▒
- +   2.76%  lto1-wpa  lto1              [.] inflate_fast                     ▒
- +   2.17%  lto1-wpa  lto1              [.] lto_output_tree(output_block*, tree▒
```

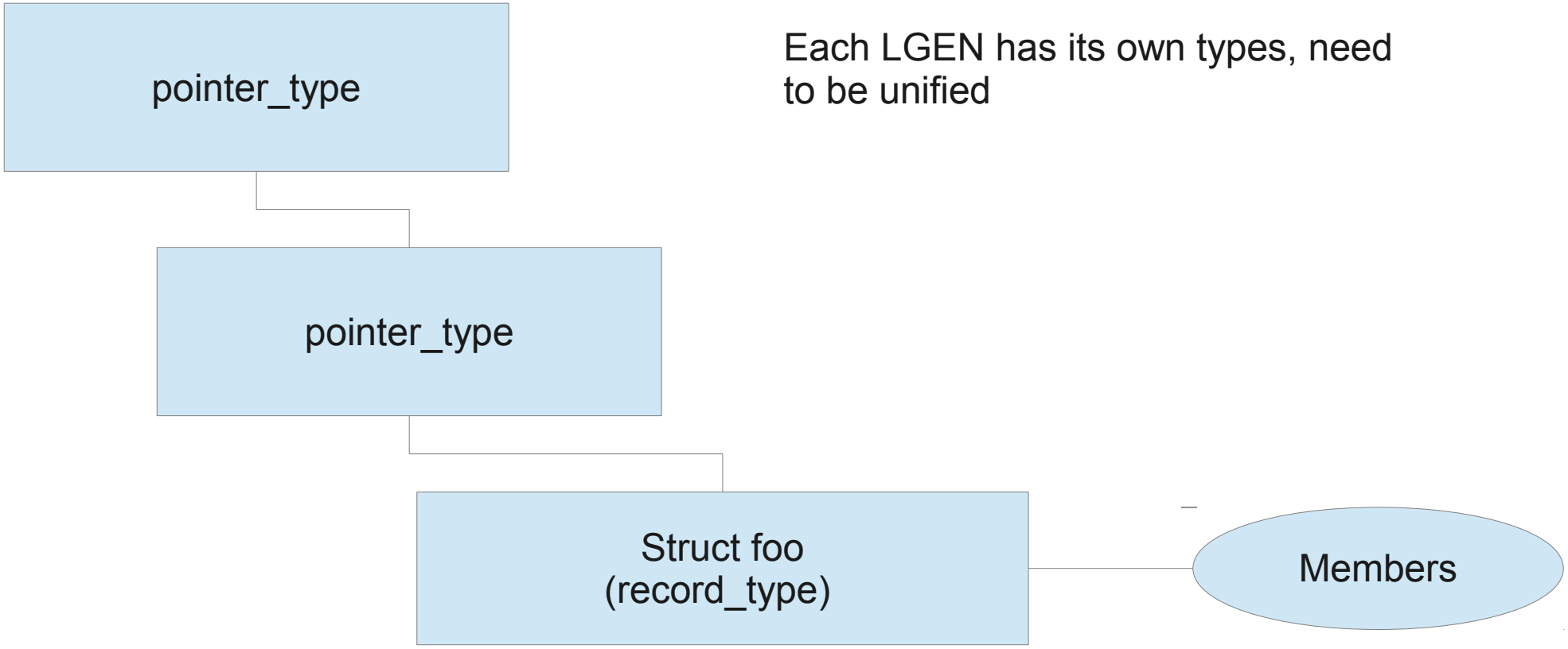
Most time spent in hashing types for type merging

Type merging

struct foo **

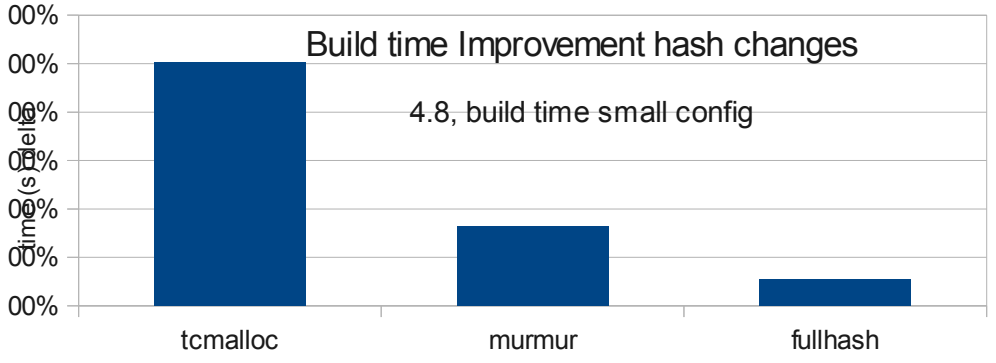
Complicated iterative hash consisting of 4 hash tables
(including two “hash cache hashes”)

Each LGEN has its own types, need to be unified



Hashing improvements

- Use tcmalloc instead of glibc malloc
  - build time -9%; peak mem +35%
- Start with large type hash tables / caches
- Use murmur for hashing / fix iterative / fix pointer hash to handle all bits
  - (collisions 90%->70%->64%)
- Splay trees for type merging (bad idea)



Possible future hash improvements

- Precompute hash in LGEN
  - And stream types out in hash order
  - Requires even larger hash tables to avoid collisions (or a good tree)
- Only merge per partition?
  - And do that in parallel
- Parallelize WPA further?

Incremental linking

- Kernel build system uses ld -r / ar extensively
- Linker merges all sections with the same name
  - Added random postfixes to LTO sections
- Still problem with pure assembler files
  - .S assembler code dropped during LTO
  - Originally attempted to fix with slim LTO
- Then fixed in Linux binutils
  - Unfortunately changes not merged into mainline binutils
  - **Use Linux binutils for kernel**

Kernel modifications (excerpt)

- Section attributes
  - const __initconst char * vs __initconst char * const
- Toplevel inline assembler
  - Need globals and visible for any symbols
- Lots of externally_visible
- Various changes for dot NUMBER symbols
- Disable LTO for some files
- Workarounds for 4.7 partitioning bugs (visible)
- Gcc-ld
- A few fixes for optimizations (rare)

Kernel changes:

108 files changed, 740 insertions(+), 313 deletions(-)

(v3.8; without most section attributes)

Debugging

- Kernel debugging: more difficult with more inlining
  - Especially with initialization functions
  - May need inline aware backtracer (“dwarf kallsyms”)

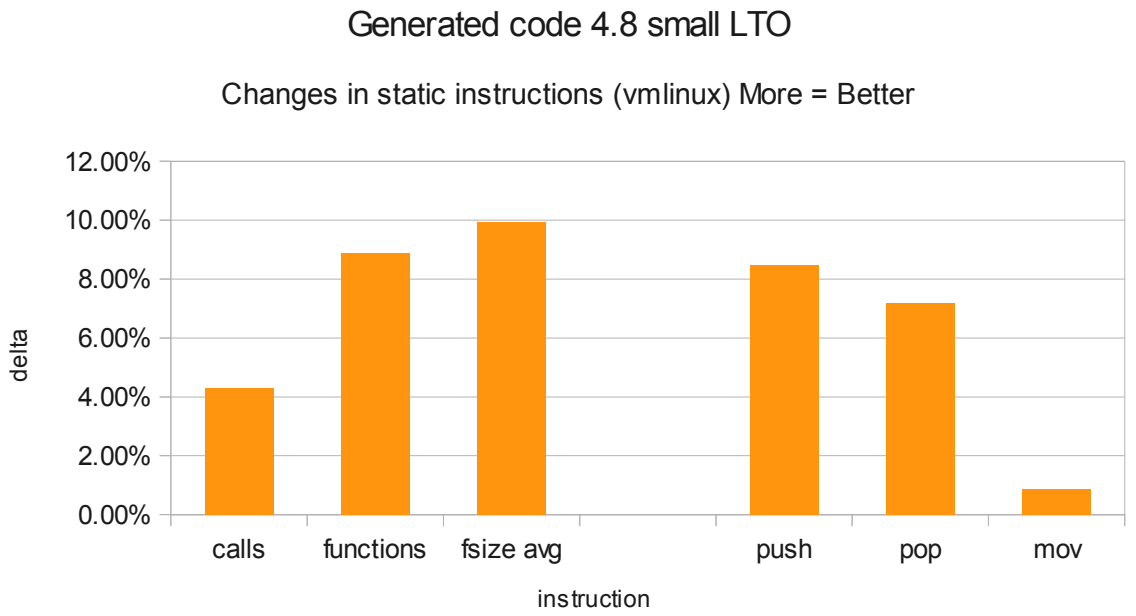
- Compiler
  - No simple test cases. Lots of regressions; Complicated delta.
  - Complex multi process
  - Hard to find right process to attach/profile
  - -dH (core dumps) are most useful

LTO has challenges for debugging

Global Optimizations (small config)

Inlining	21k new inlines between files 13k unique
Scalar replacement	No change
Partial Inline	+16%
Constant Propagation	+40%
Cloning for constant propagation (CONFIG_LTO_CLONE)	266->1052 const clones +2% text (~200k)

Static instruction changes



Runtime

- Runtime
  - Mixed results

LKP	dbench	tbench	AIM7	OLTP
Sandy Bridge	0%	0%	+3%	0%
Nehalem	0%	0%	+4%	+6%

- Likely need some more optimizations
  - Open: Atom testing (more sensitive to bad code)

Text size

- Text size
  - Slight increase for large configs, but lower for small configs
    - small config +3%
    - allno shrinks -50k
  - The smaller the kernel the less infrastructure is used. But EXPORT_SYMBOL defeats it again.

ARM embedded improvements (Tim Bird)

```
 vmlinux.non-lto  =>  vmlinux.lto
       baseline   other   change  percent
       --------   -----   ------  -------
 text:  5242000  4869340  -372660  -7%
 data:   490184   476200   -13984  -2%
  bss:   121824   122112      288   0%
total:  5854008  5467652  -386356  -6%

Kernel:                         non-lto         lto
---------------------------     -------         --------
time for full build:            1m5.87s         3m22s
time for incremental build:     0m5.24s         2m12s
kernel image size:              5854008         5467652
compressed uImage size:         2849920         2694224
system meminfo MemTotal:        17804 kB        18188 kB
system meminfo MemFree:         11020 kB        11260 kB
boot time:                      2.466369        2.414428
```

gcc LTO improvements wish list

- Per file options (partial patch)
- Fix passing through of options
- Standard gcc-ld
- Don't error for section mismatches
- Better procedure for LTO error reporting
- Better messages for type mismatches
- No .XXXX postfixes
- -fno-toplevel-reorder by symbol?
- Automatic procedure for externally_visible discovery?
- Better syntax for “symbol visible to toplevel assembler”

Status

- Tested on x86, ARM, MIPS
- Uptodate with Linux 3.8
- Some options disabled (function-trace, gcov, modversions)

References

- LTO kernel tree

  http://github.com/andikleen/linux-misc
  - Branch lto-3.x (x increasing)
- Linux binutils
  http://www.kernel.org/pub/linux/devel/binutils/
- Tcmalloc http://code.google.com/p/gperftools/

- andi@firstfloor.org http://halobates.de

Backup

Future kernel improvements

- Avoid multi-link
- Use constructors to allow disabling -fno-toplevel-reorder
- Support new option to disable instrument_function

Quick HOWTO for LTO kernel

- Get git://github.com/andikleen/linux-misc lto-3.x tree
- Need gcc 4.7+ set up for LTO / Linux binutils
- Enable CONFIG_LTO
  - Make sure FUNCTION_PROFILER et.al./MODVERSIONS are disabled
- make CC=compiler LD=linux-ld AR=gcc-ld -jX
  - Check LTO does not get disabled
- See Documentation/lto-build for more details