

Building openSUSE with link-time optimizations

Jan Hubička and Martin Liška
SUSElabs
jh@suse.cz, mliska@suse.cz

Outlilne

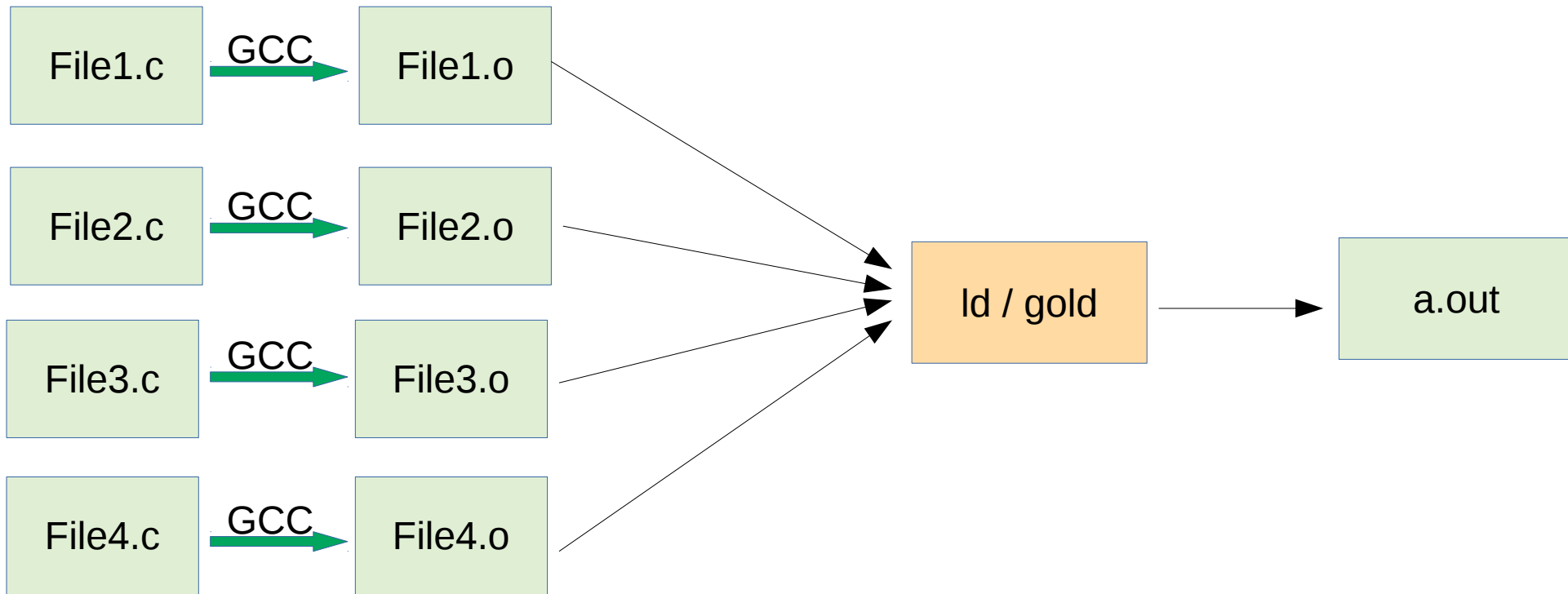
- What is link-time optimization?
- Link-time optimization and GCC
- Benchmarks
- Can we build openSUSE with link-time optimization by default?



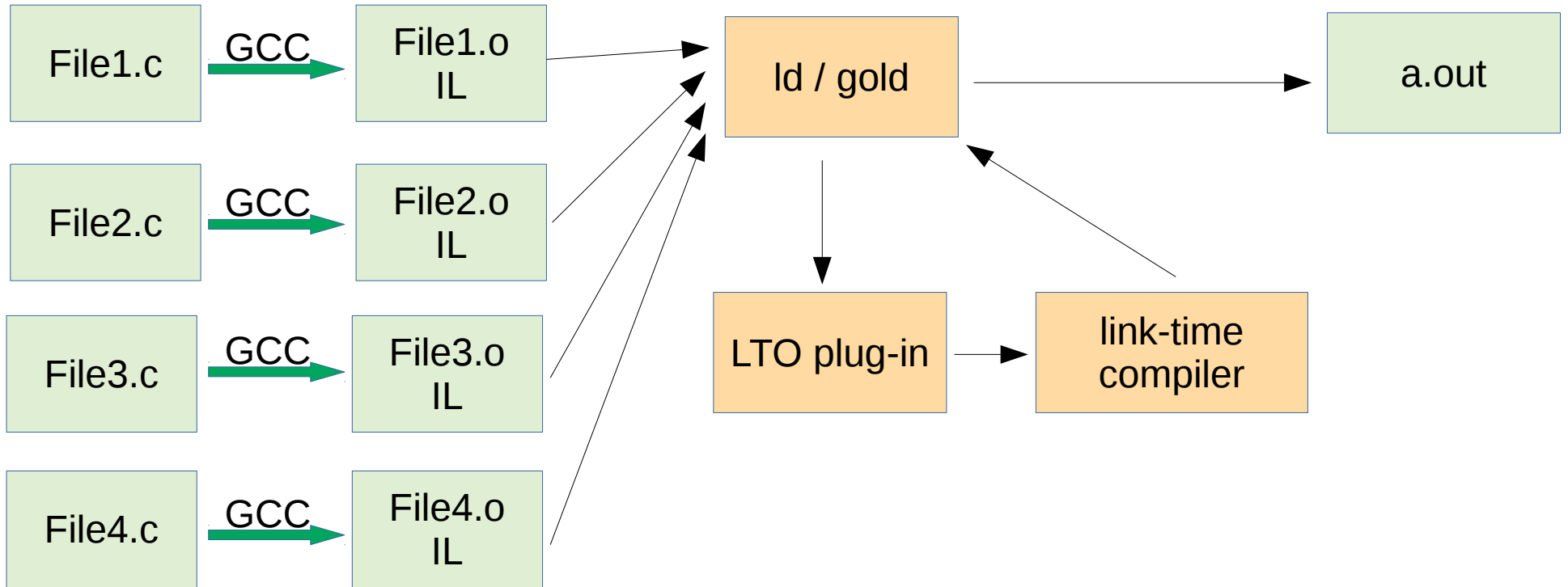


What is link-time
optimization?

Per-file compilation



Link-time optimization



Benefits of LTO

- Symbol promotion
(from linker's resolution data most symbols become “static”)
- Cross-module inlining, constant propagation
- Aggressive unreachable code removal
- Profile propagation
- EH optimization (propagation of “nothrow”)
- Identical code folding
- Optimized code layout
- and more :)



Problems of LTO

- Whole toolchain has to be restructured
- Slow compile-edit cycle
- Harder bugreports
(often whole program needed to reproduce issue)
- Not 100% transparent to user, but in most cases all one needs to do is to add -flto



Link-time optimization and GCC

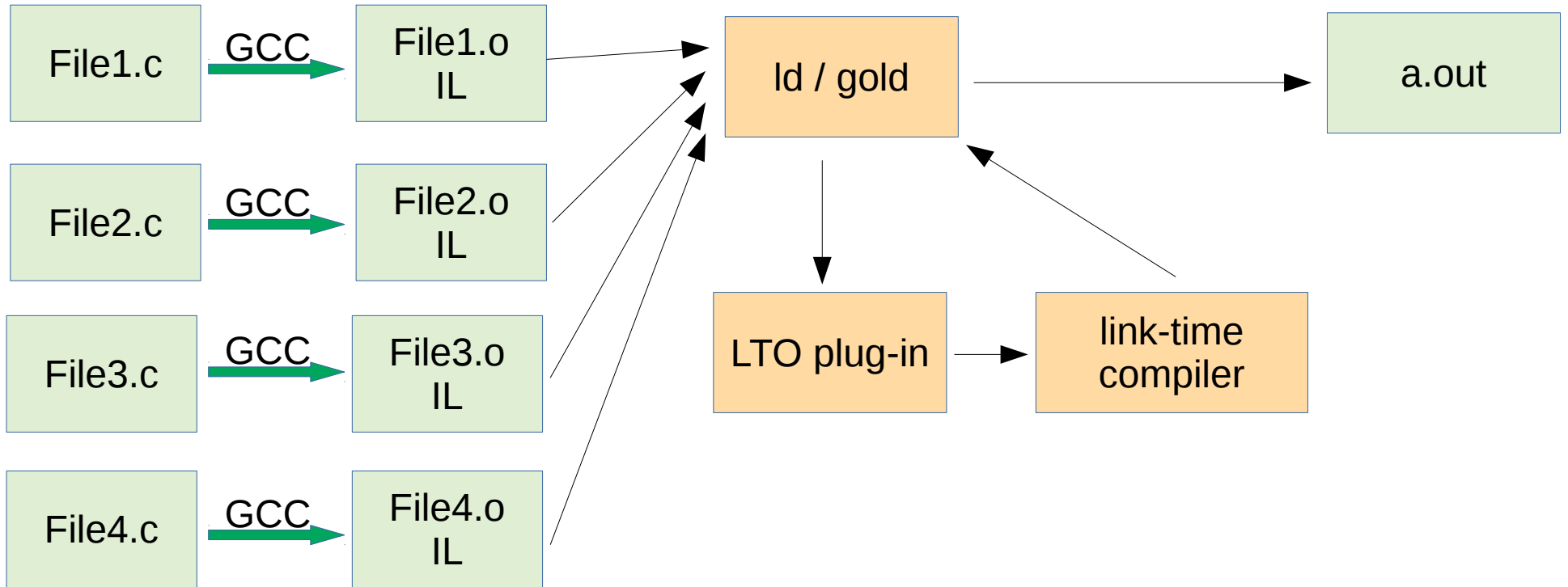
The background features a large teal shape on the left and a green shape on the right, separated by a white diagonal line. The teal shape has a white arrow-like cutout on its right side that points towards the green shape.

Modernizing GCC (LTO perspective)

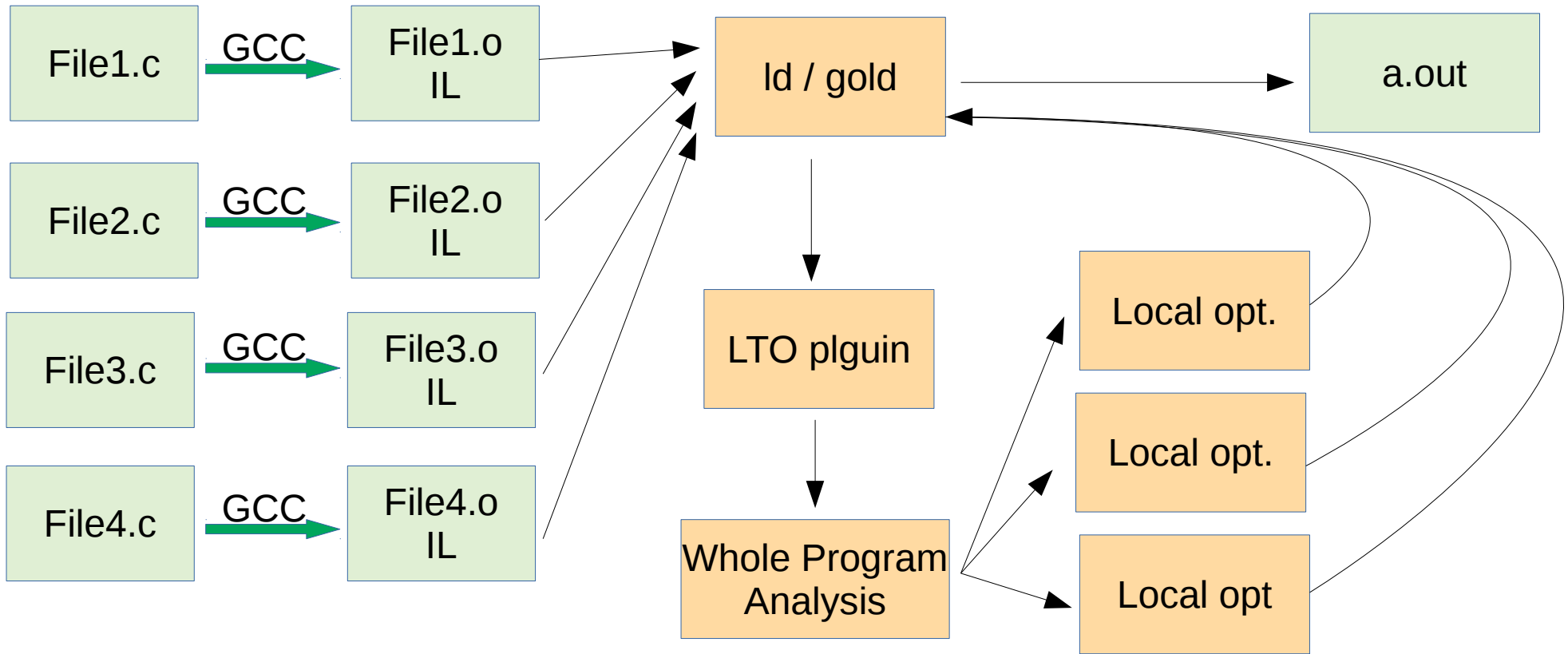
- **1999 (GCC 2.95):** Function-at-a-time
- **2001 (GCC 3.0):** New inliner (first high-level opt . in gcc)
- **2004 (GCC 3.4):** Unit-at-a-time; intermodule compilation for C; Inter-procedural optimization framework
- **2005 (GCC 4.0):** New SSA optimization framework
- **2006 (GCC 4.1):** Inter-procedural optimizations: profile guided inlining, pure/const discovery, mod/ref, inter-procedural constant propagation, --fwhole-program -combine
- **2008 (GCC 4.4):** Inter-procedural optimization on SSA; early optimization and inlining.
- **2010 (GCC 4.5):** Basic LTO framework (5 years in development)
- **2011 (GCC 4.6):** WHOPR (parallel link-time optimization); Firefox builds



Link-time optimization



Parallelized Link-time optimization (WHOPR)



Modernizing GCC (LTO perspective)

- **2012 (GCC 4.7.0):** Memory use optimizations, new inliner heuristics, new inter-procedural constant propagation with cloning
- **2013 (GCC 4.8.0):** symbol table; propagation of values passed through aggregates
- **2014 (GCC 4.9.0):** slim LTO objects by default; on demand loading of functions; devirtualization pass; feedback directed code layout
- **2015 (GCC 5):** Identical Code Folding; COMDAT optimization; One Definition Rule for C++; alignment propagation; correct command line options handling with LTO
- **2016 (GCC 6):** Linker-plugin now detects type of output binary. C&Fortran type merging. Better alias analysis
- **2017 (GCC 7):** Inter-procedural value range propagation; bitwise propagation
- **2018 (GCC 8):** Early debug info. Profile representation rewrite. Function splitting now by default. Reworked runtime estimation; Malloc attribute propagation



GCC optimization pipeline

Compile time

Parser

IL generation

Early opts:

- Early Inliner
- Constant prop.
- Forward prop.
- Jump threading
- Scalar repl. of aggr.
- Alias analysis
- Redundancy ellim.
- Dead store ellim.
- Dead code ellim.
- Tail recursion
- Switch conversion
- pure/const/nothrow
- EH optimization
- Profile guessing

IP analysis streaming out

Link-time serial

Streaming in symbols, types and declarations & link

Inter-procedural (whole program) Opts:

- Dead symbol ellim.
- Symbol promotion
- profile analysis
- Identical code folding
- devirtualization
- Constant propagation
- const/destr merging
- Inlining
- pure/const/nothrow
- mod/ref
- comdat

Partitioning streaming out

Link-time parallel

Stream in and apply transformations

High level opts:

- Constant prop.
- Complete unroll
- Forward prop.
- Alias analysis
- Return slot opt.
- Redundancy ellim.
- Jump threading
- Dead code ellim.
- Conditional store ellim.
- Copy prop.
- If combine
- Tail recursion
- Copy loop headers
- Scalar repl. of aggr.
- Dead store ellim.
- Dead code ellim.
- Reassociation
- Sincos, bswap opt.
- Loop invariant motion
- Partial redundancy ellim.
- Loop splitting
- Unroll and jam
- Loop dsitribution
- Loop interchange ...

Low level opts:

- Common subexpression ellim.
- Forward propagation
- Copy propagation
- Partial Redundancy Ellim.
- Code hoisting
- Copy propagation
- Store motion
- If conversion
- Loop invariant motion
- Loop unrolling
- Doloop optimization
- Web construction
- Copy propagation
- Common subexpression ellim.
- Dead store ellim.
- Instruction combine
- Function partitioning
- Instruction splitting
- Live range shrinking
- Scheduling
- Register allocation
- Global common subexpr. Ellim.
- Shrink wrapping
- Stack adjustment opt.
- Register renaming
- Constant prop.
- Code reordering
- Scheduling
- X87 register stack
- Code/data alignment
- Machine dependent reorg.
- Code output

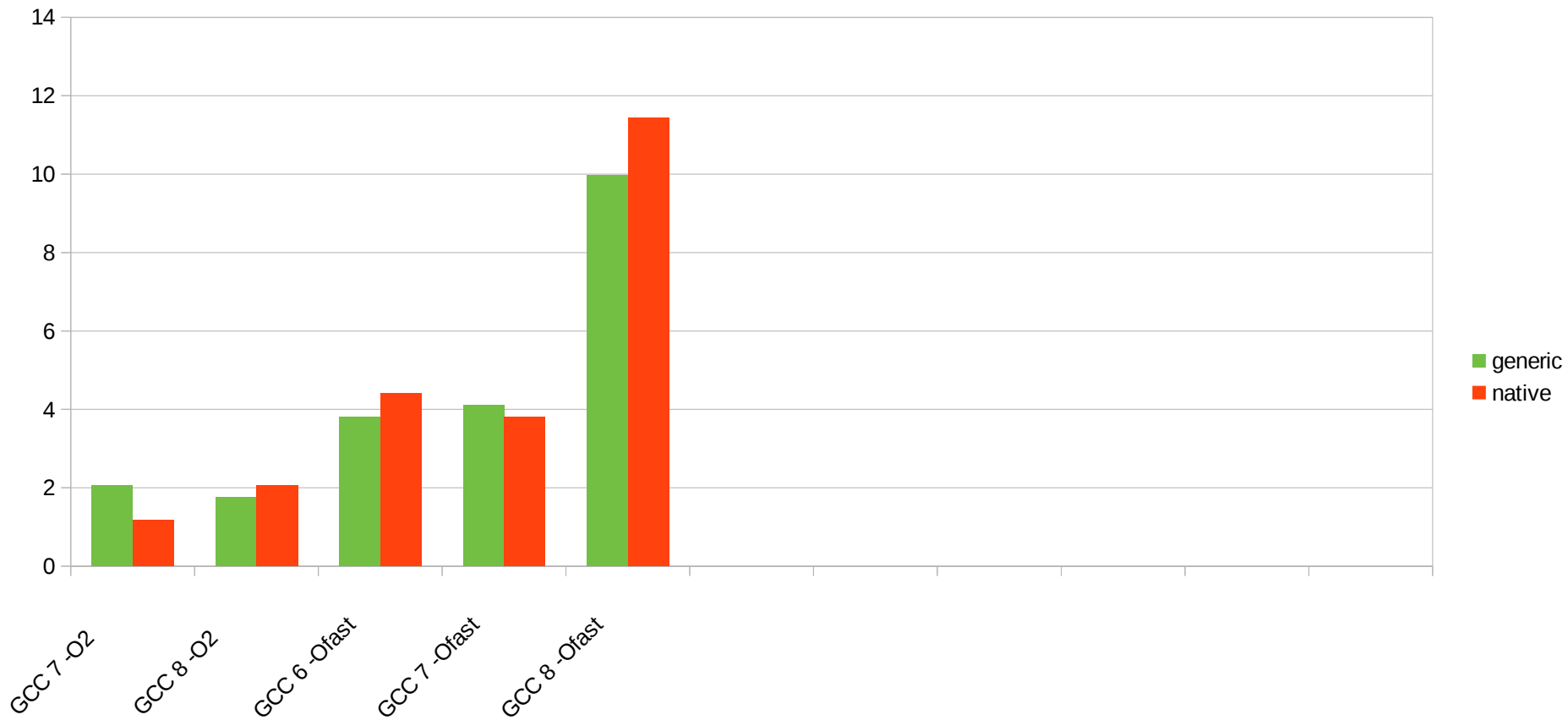
Benchmarks

The background features abstract geometric shapes. A large teal shape occupies the left and top portions, while a green shape is on the right. A white diagonal line separates the teal and green areas, creating a sense of depth and movement.

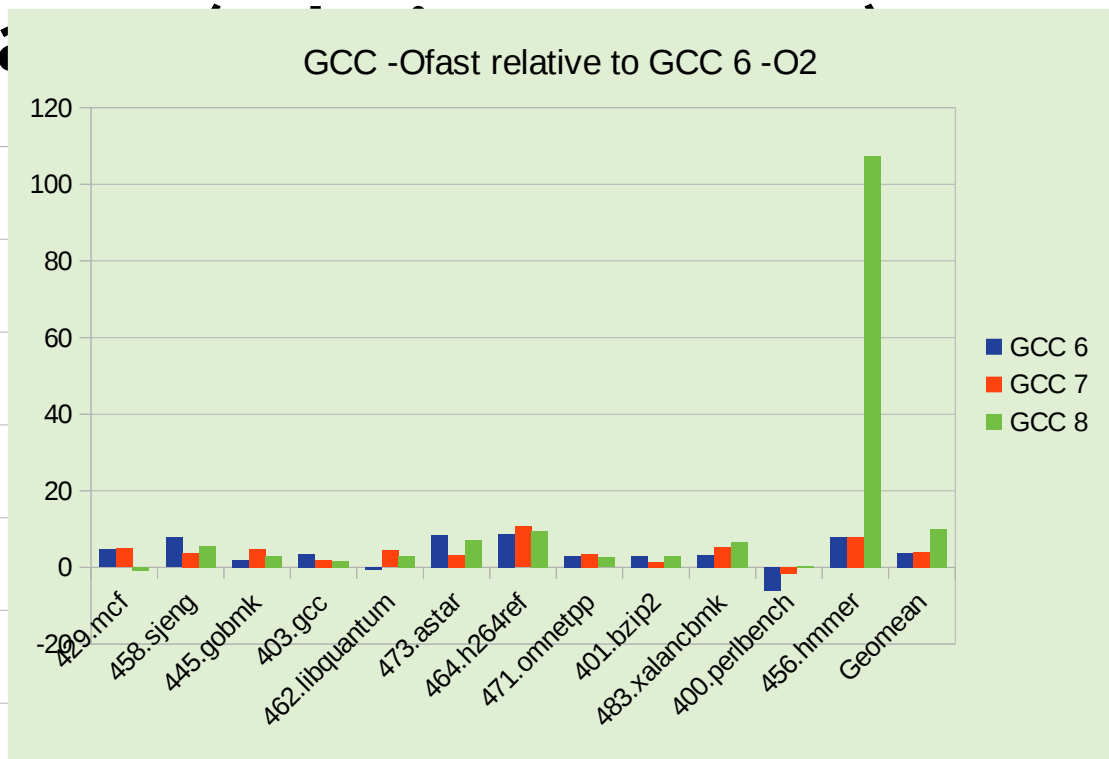
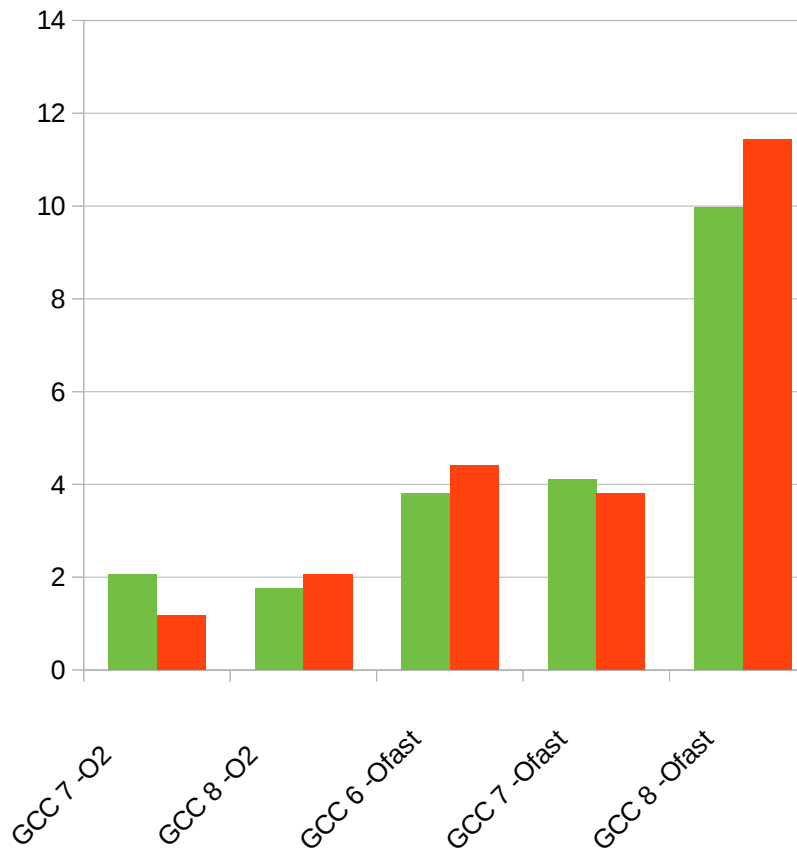
SPECint 2006 performance (relative to GCC 6)



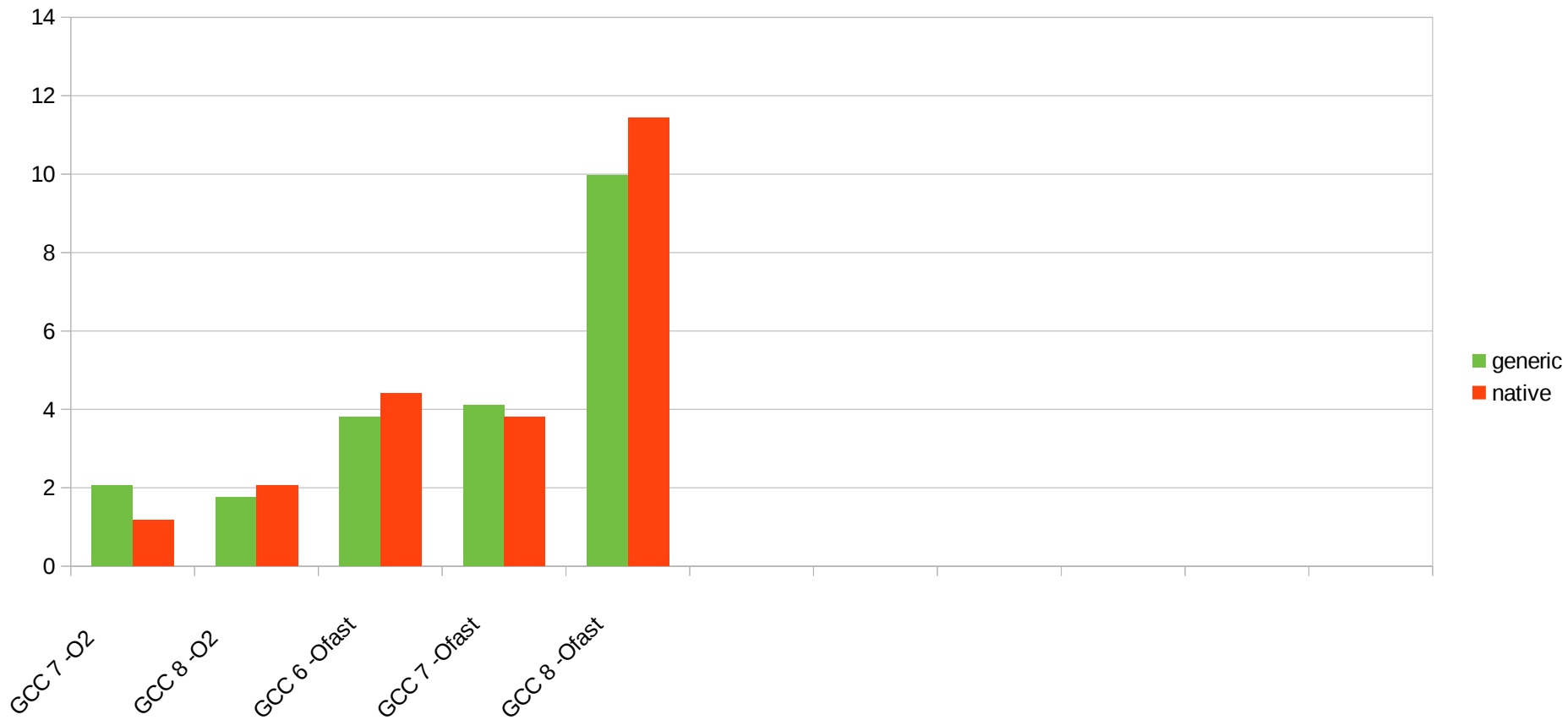
SPECint 2006 performance (relative to GCC 6)



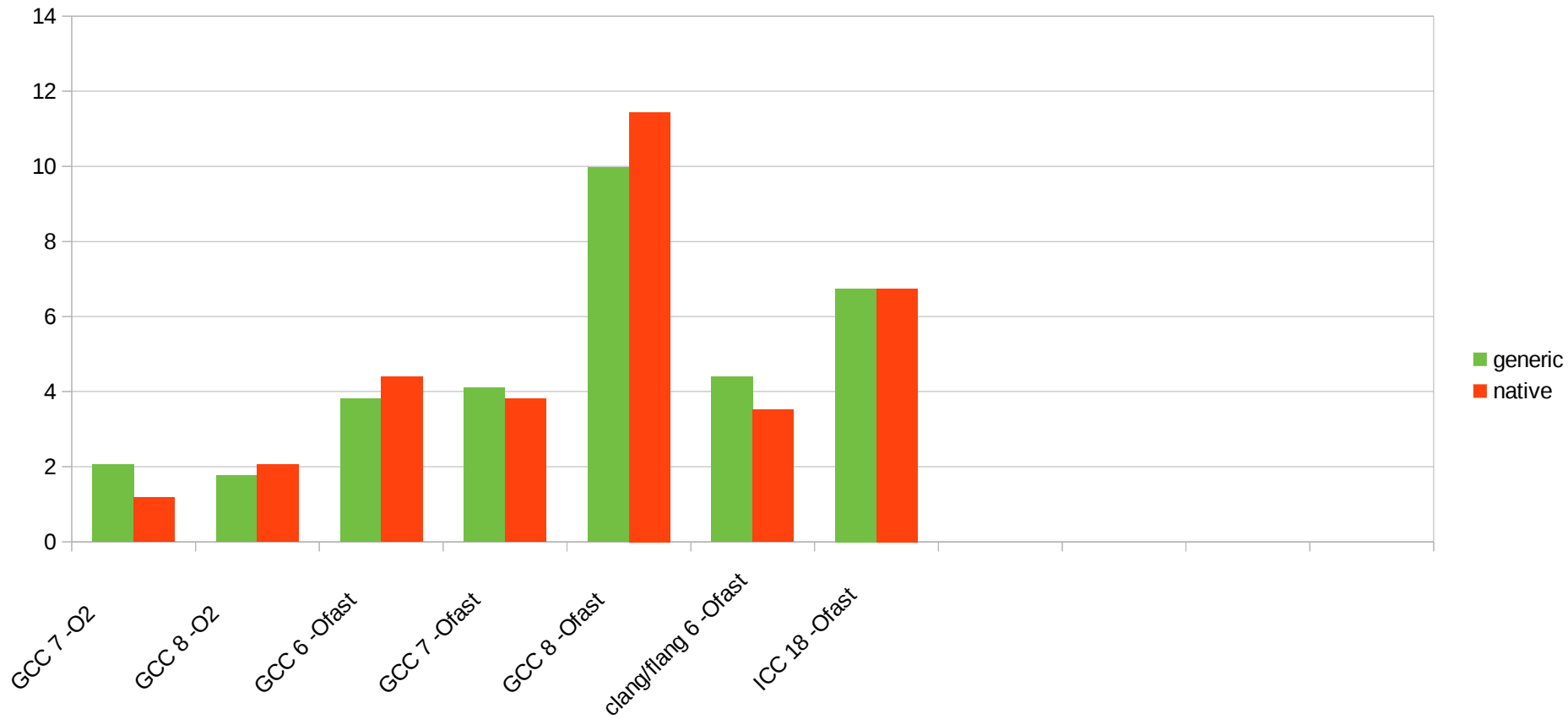
SPECint 2006 performance



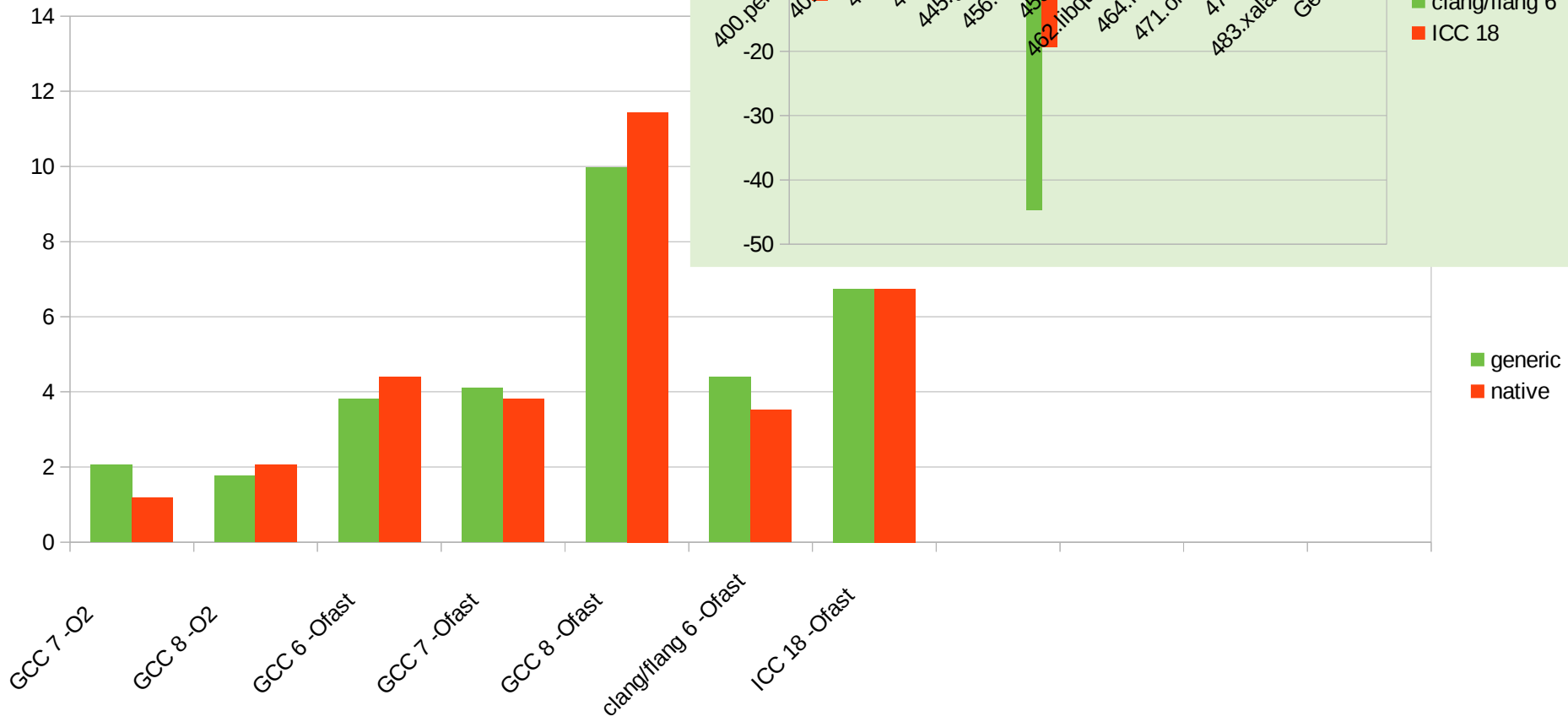
SPECint 2006 performance (relative to GCC 6)



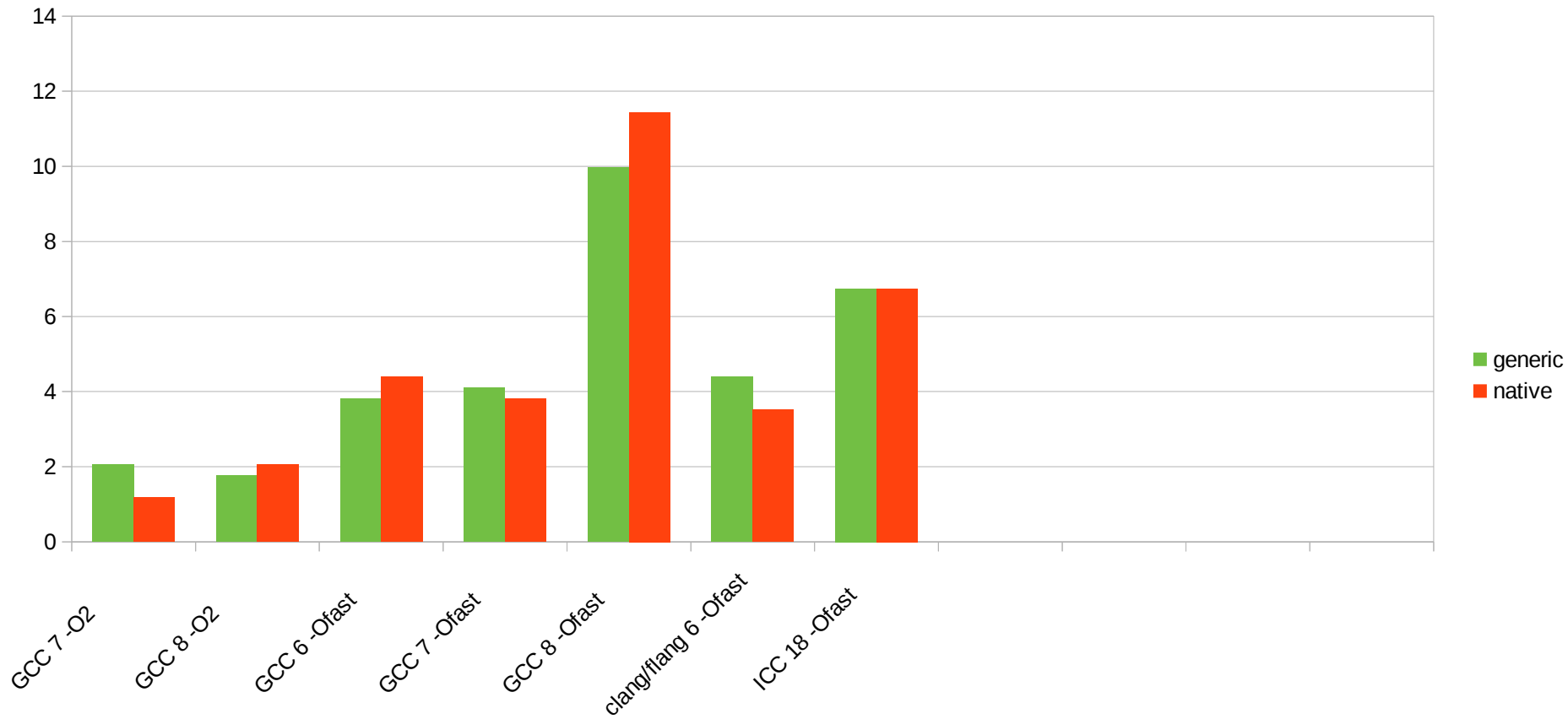
SPECint 2006 performance (relative to GCC 6)



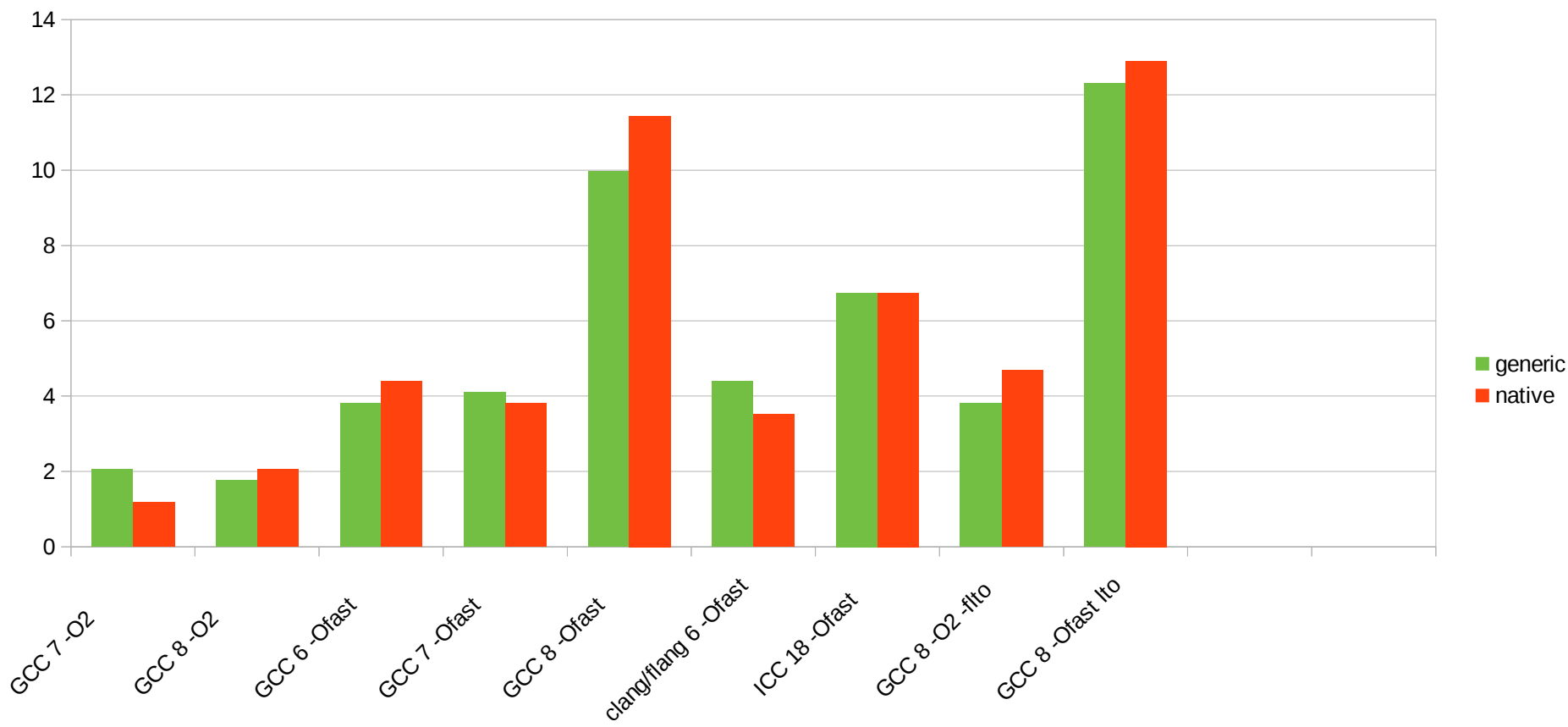
SPECint 2006 perform



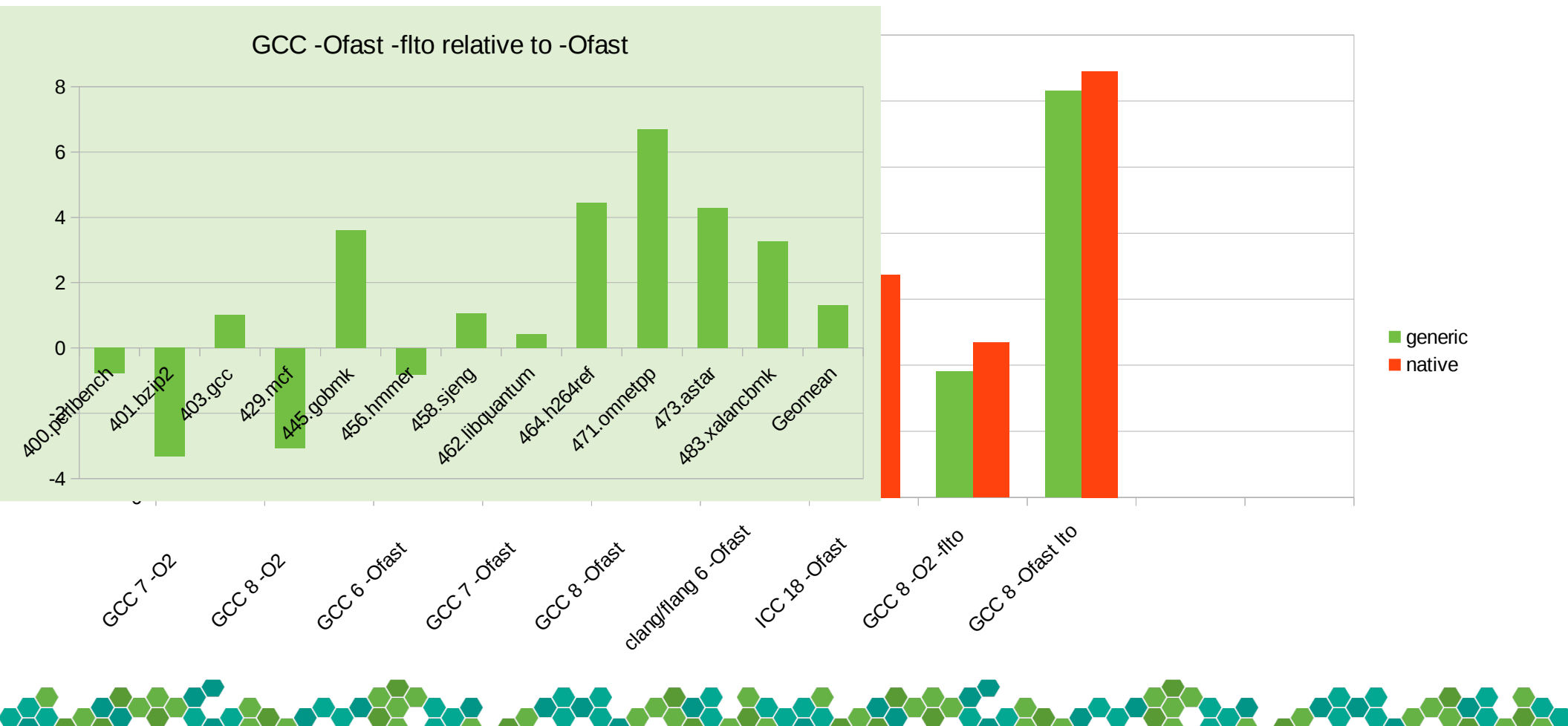
SPECint 2006 performance (relative to GCC 6)



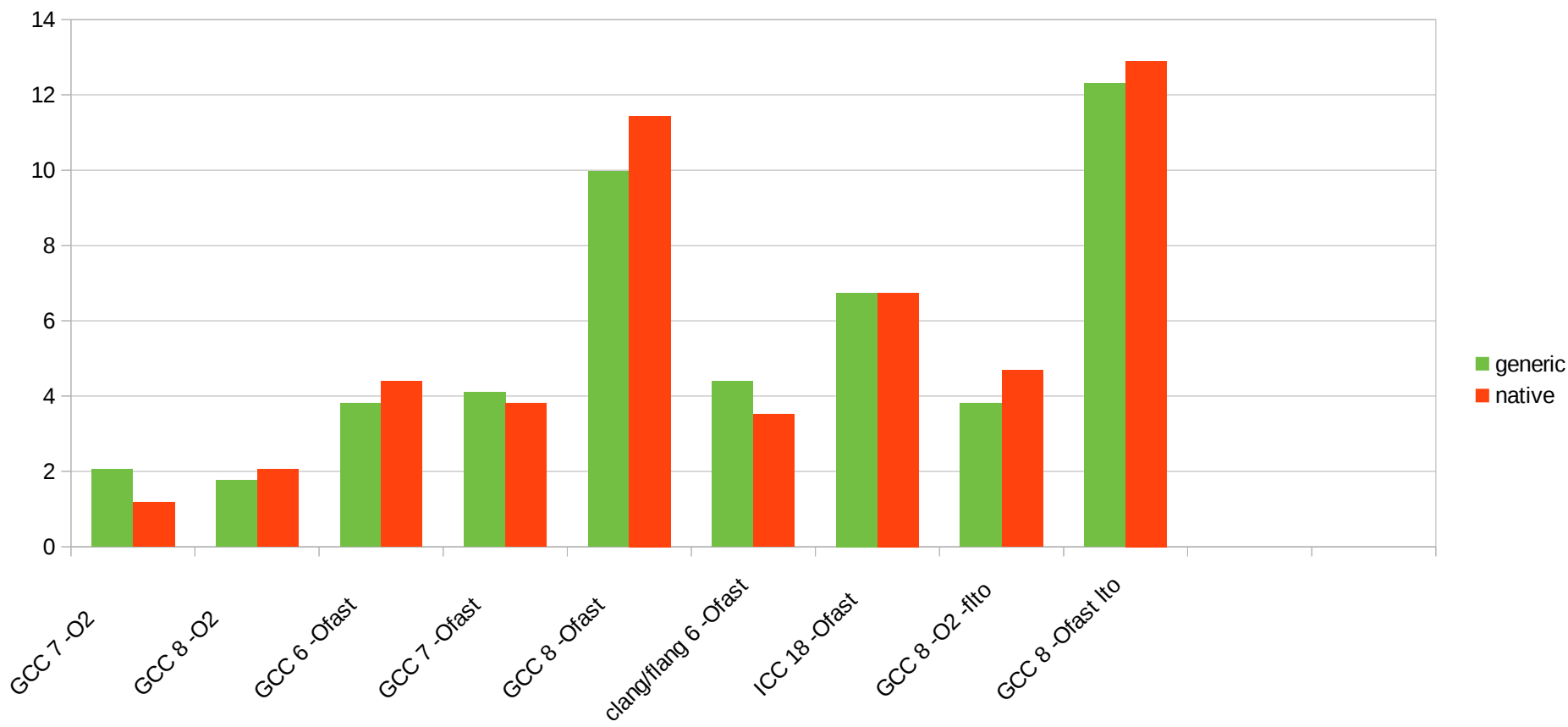
SPECint 2006 performance (relative to GCC 6)



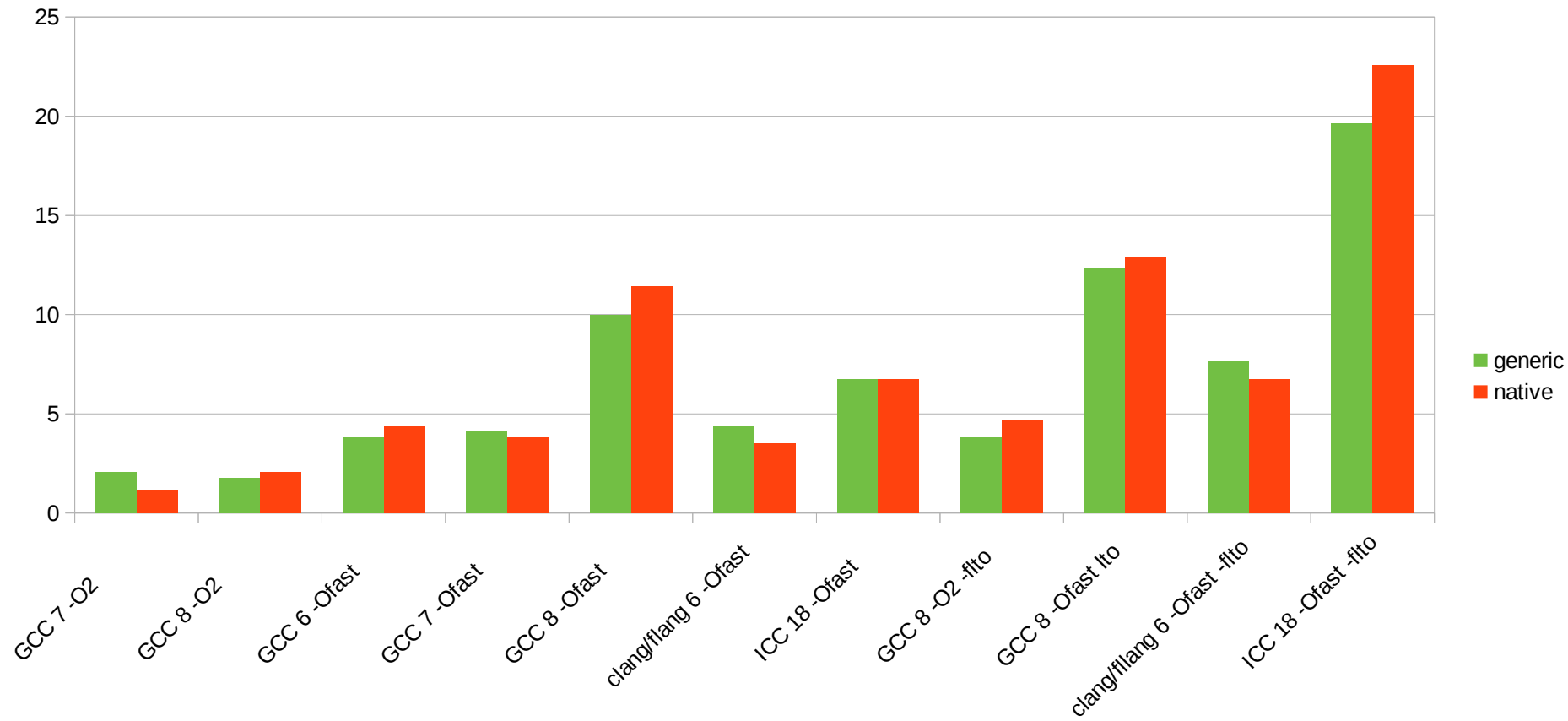
SPECint 2006 performance (relative to GCC 6)



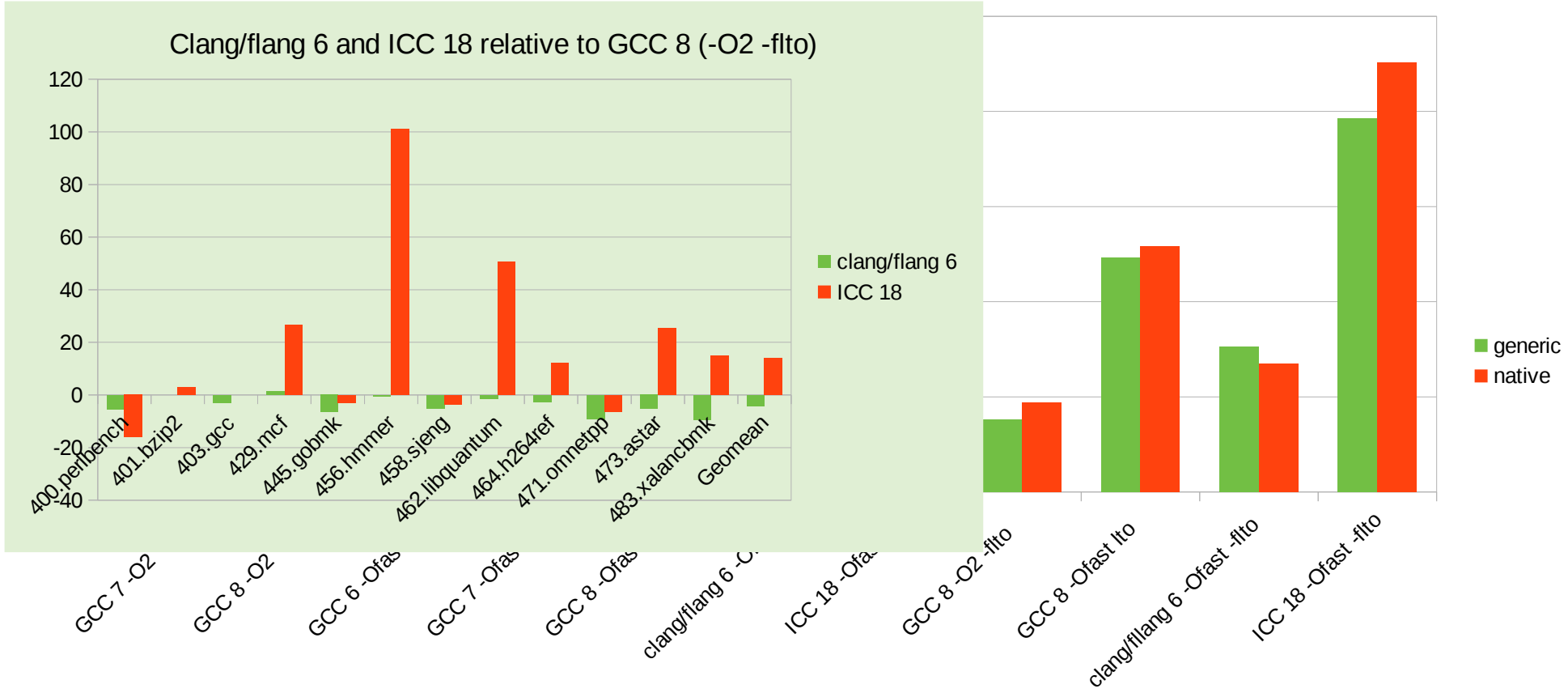
SPECint 2006 performance (relative to GCC 6)



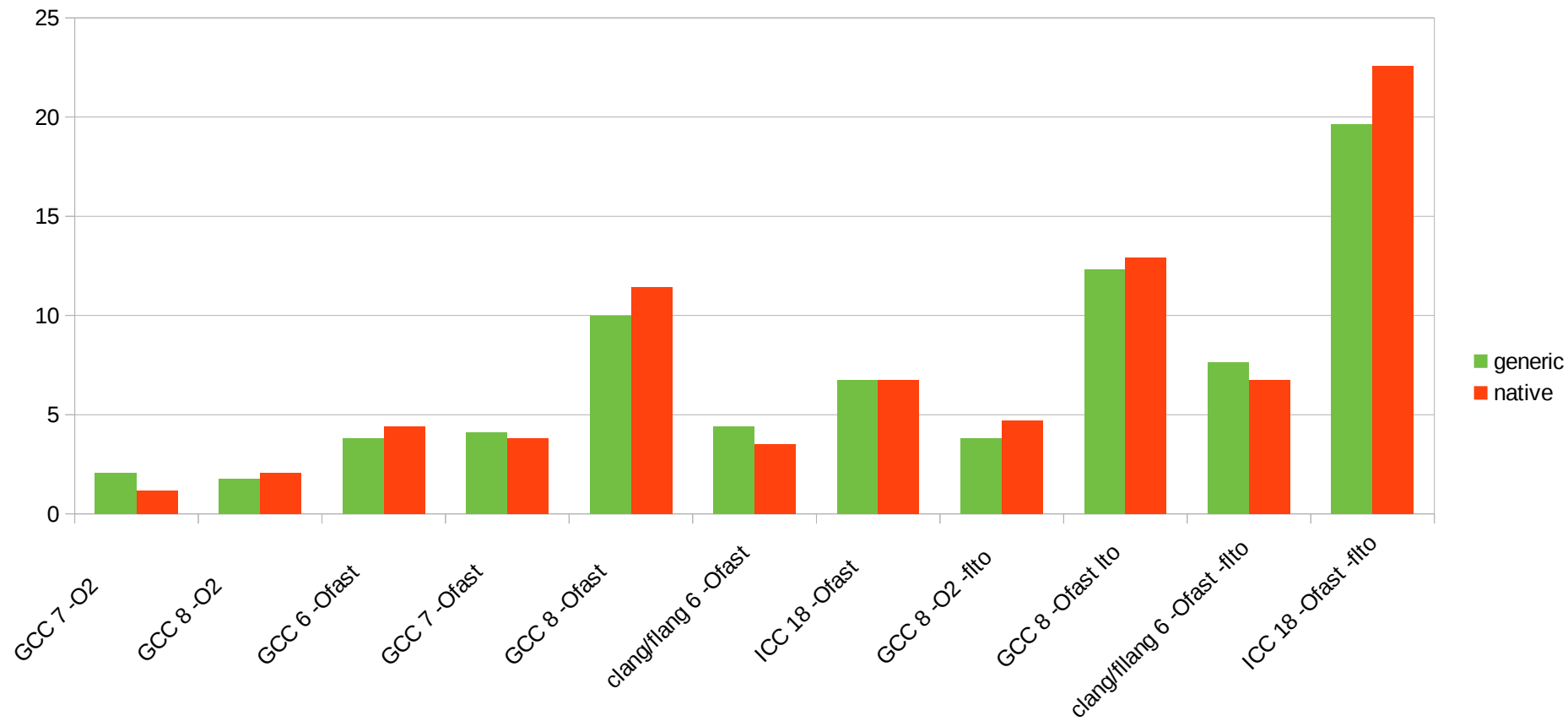
SPECint 2006 performance (relative to GCC 6)



SPECint 2006 performance (relative to GCC 6)

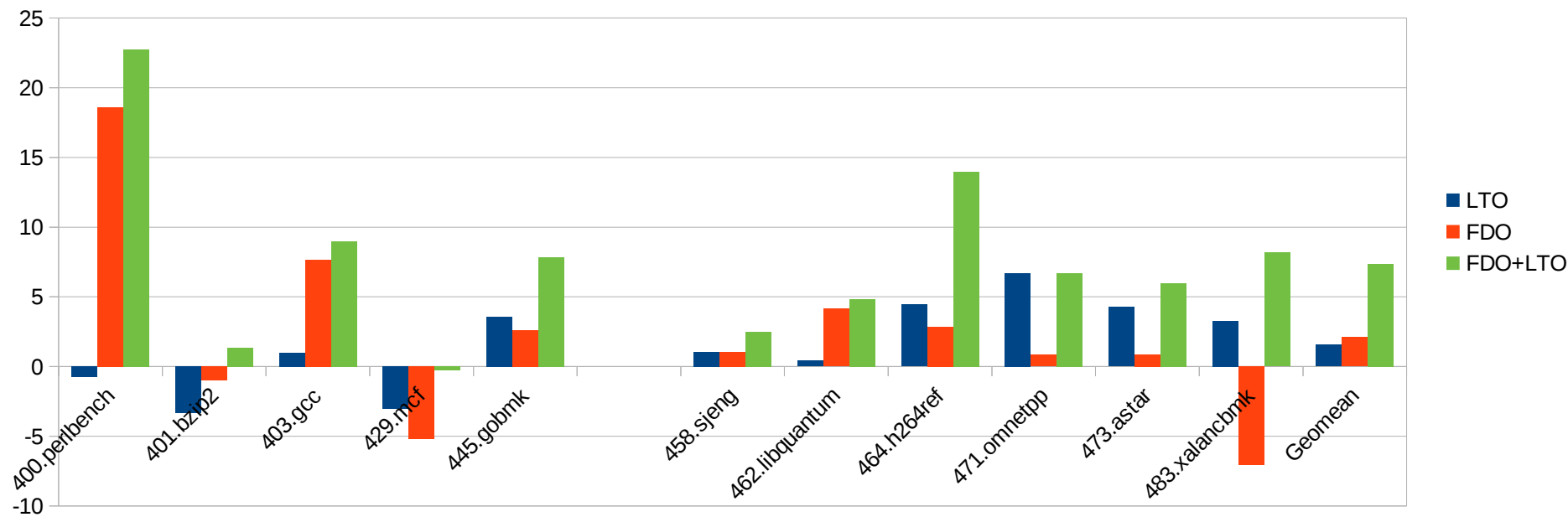


SPECint 2006 performance (relative to GCC 6)



Profile feedback & LTO works well together

Performance relative to GCC 8 -Ofast



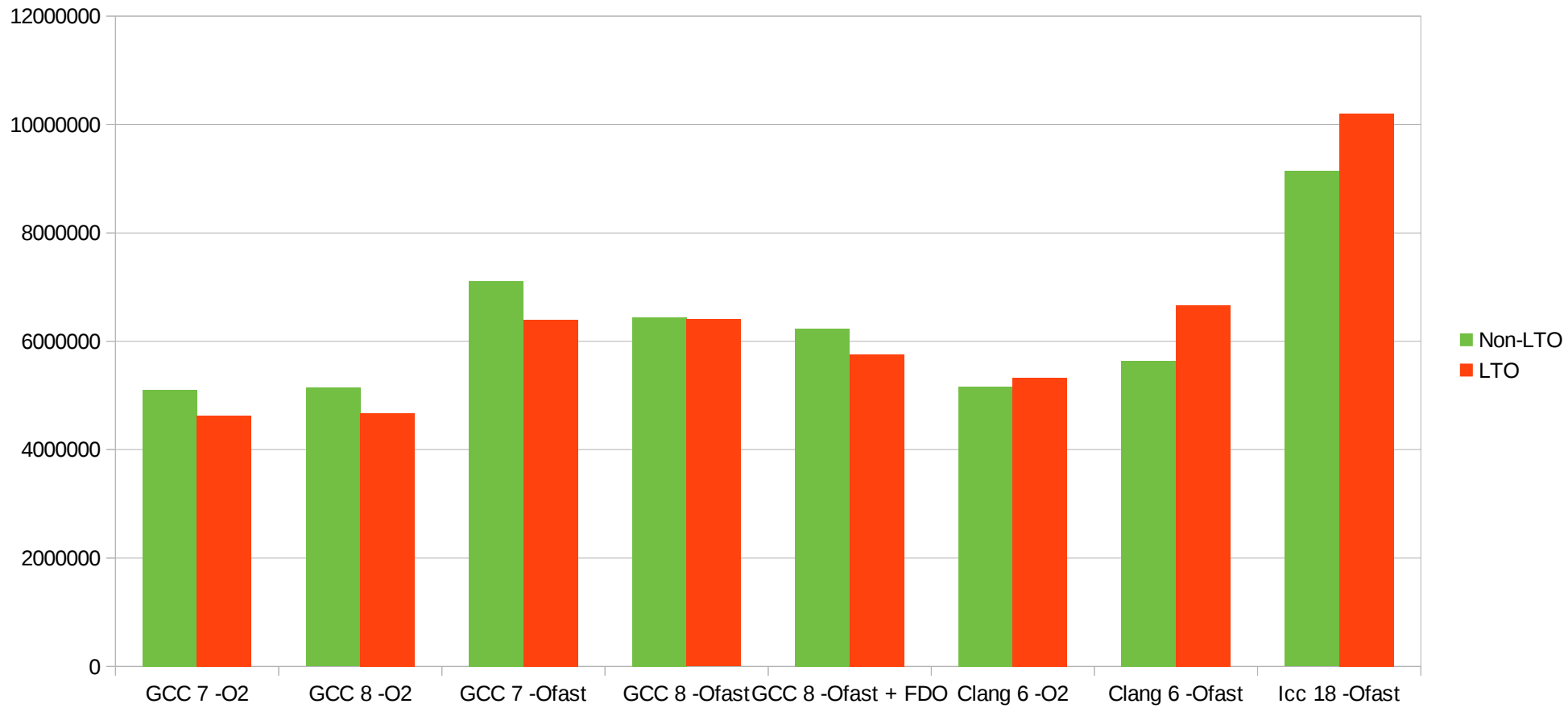
Hmmer benchmark has problems; was excluded.

To build with profile feedback often you can use:

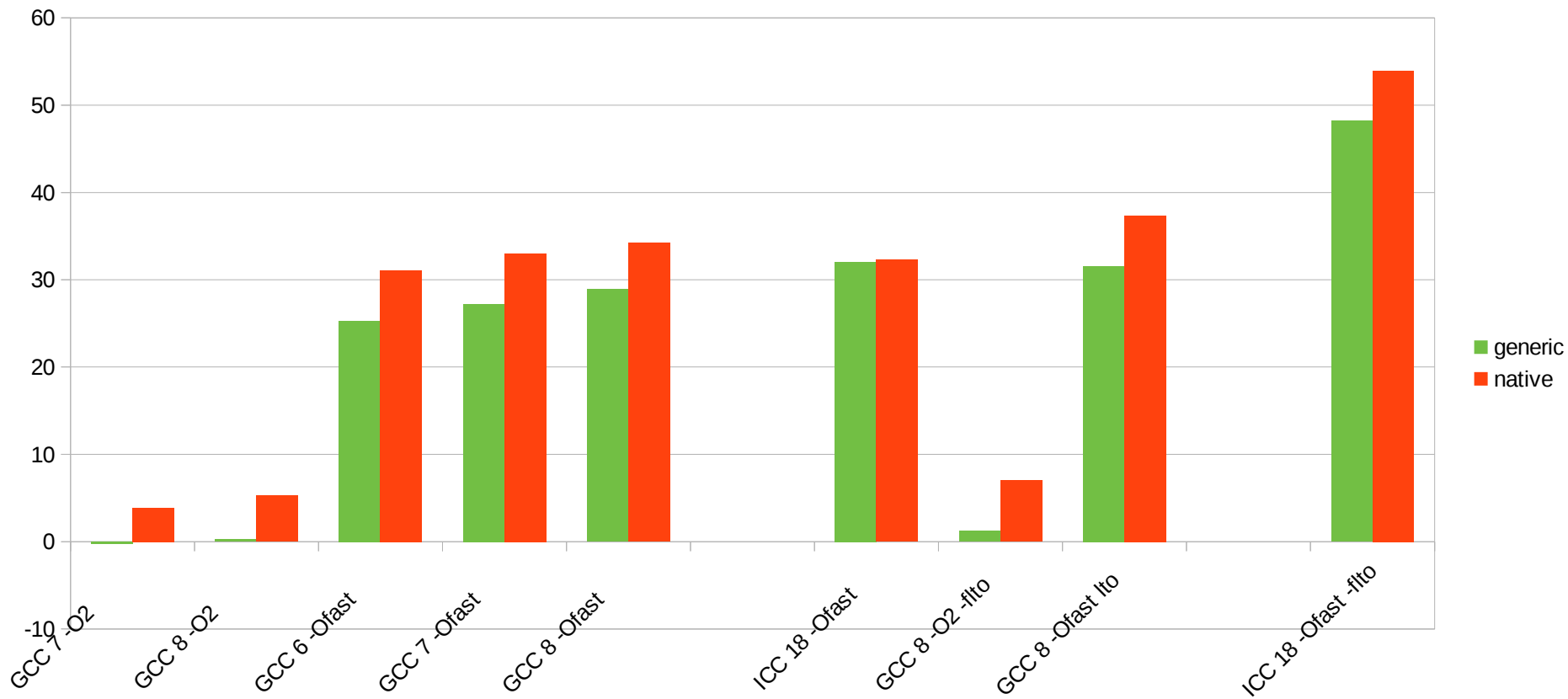
```
./configure ; CFLAGS="-O2 -fprofile-generate" make ; make check ; make clean ; CFLAGS="-O2 -fprofile-use" make
```



SPECint 2006 code size

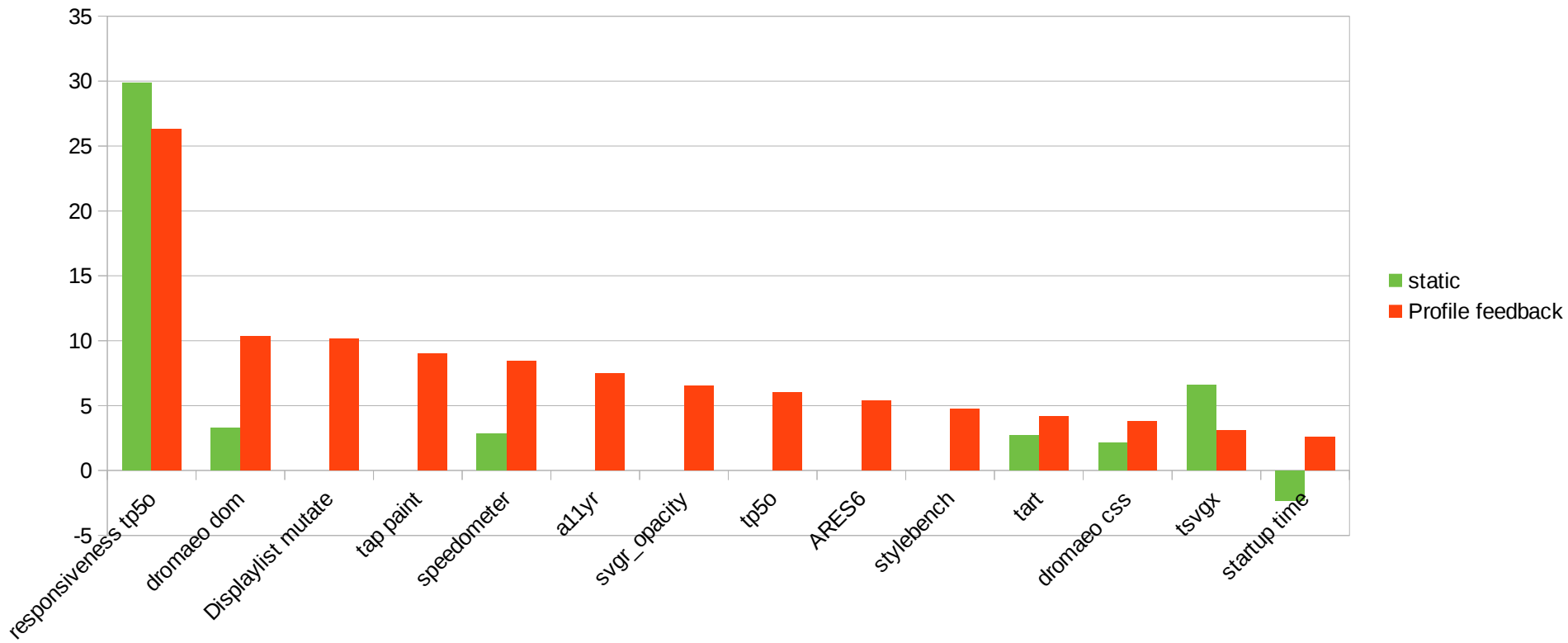


SPECfp 2006 performance (relative to GCC 6)

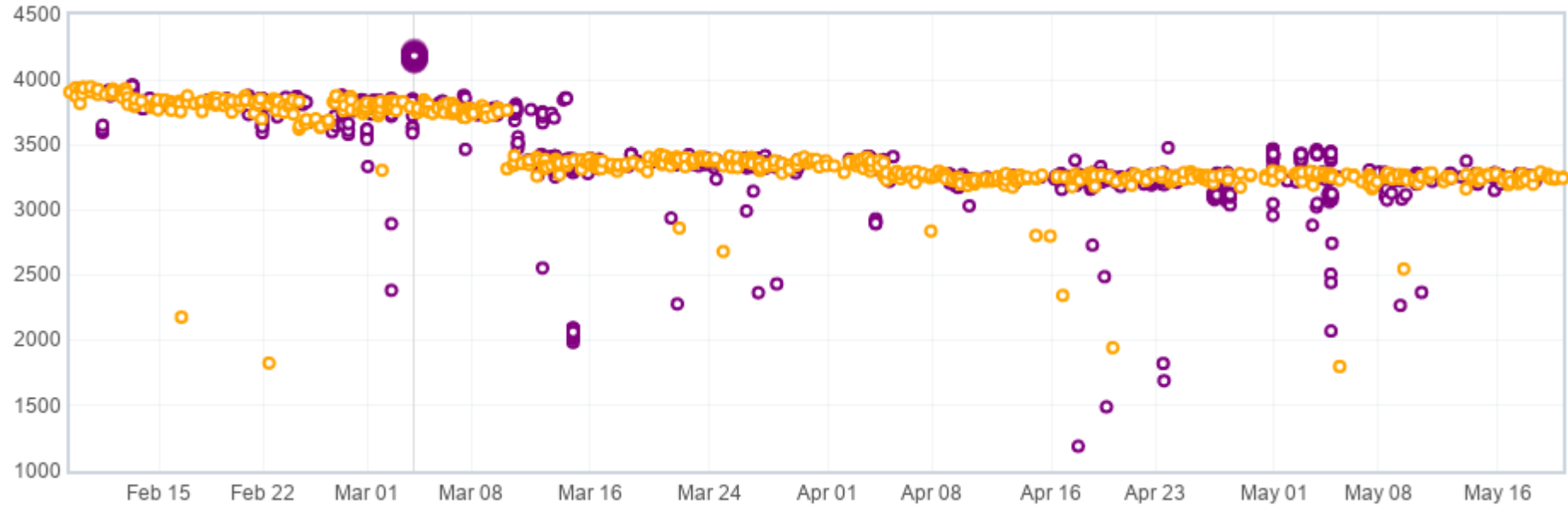


Firefox performance summary

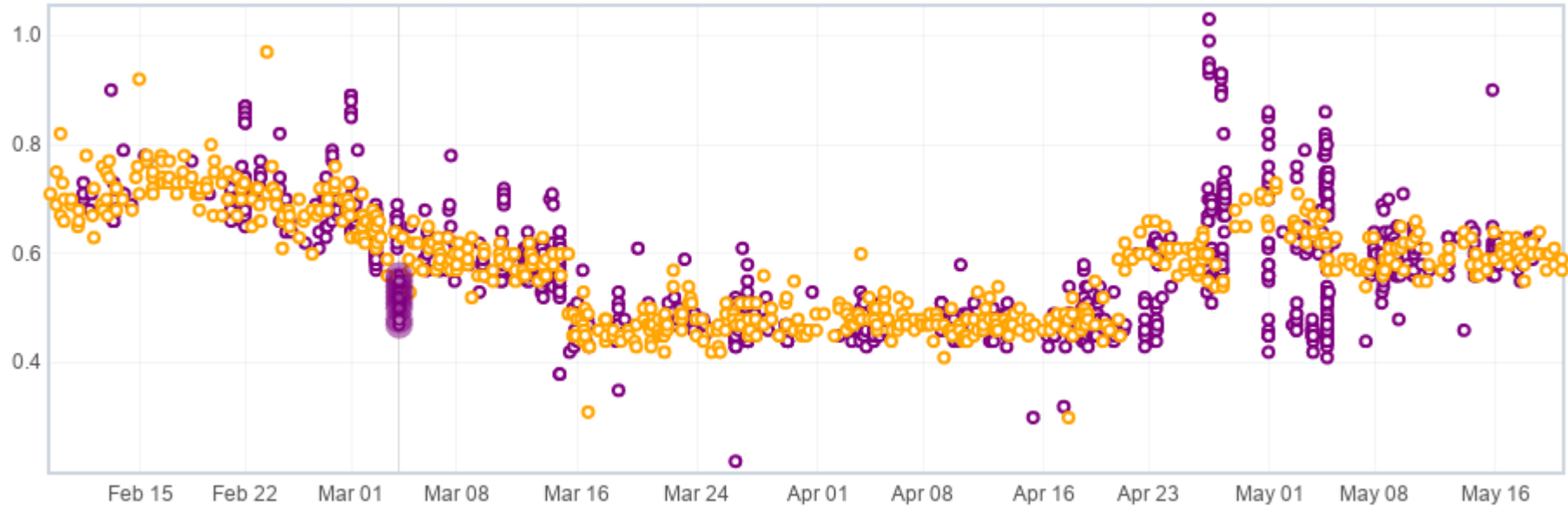
Firefox performance relative to non-LTO build



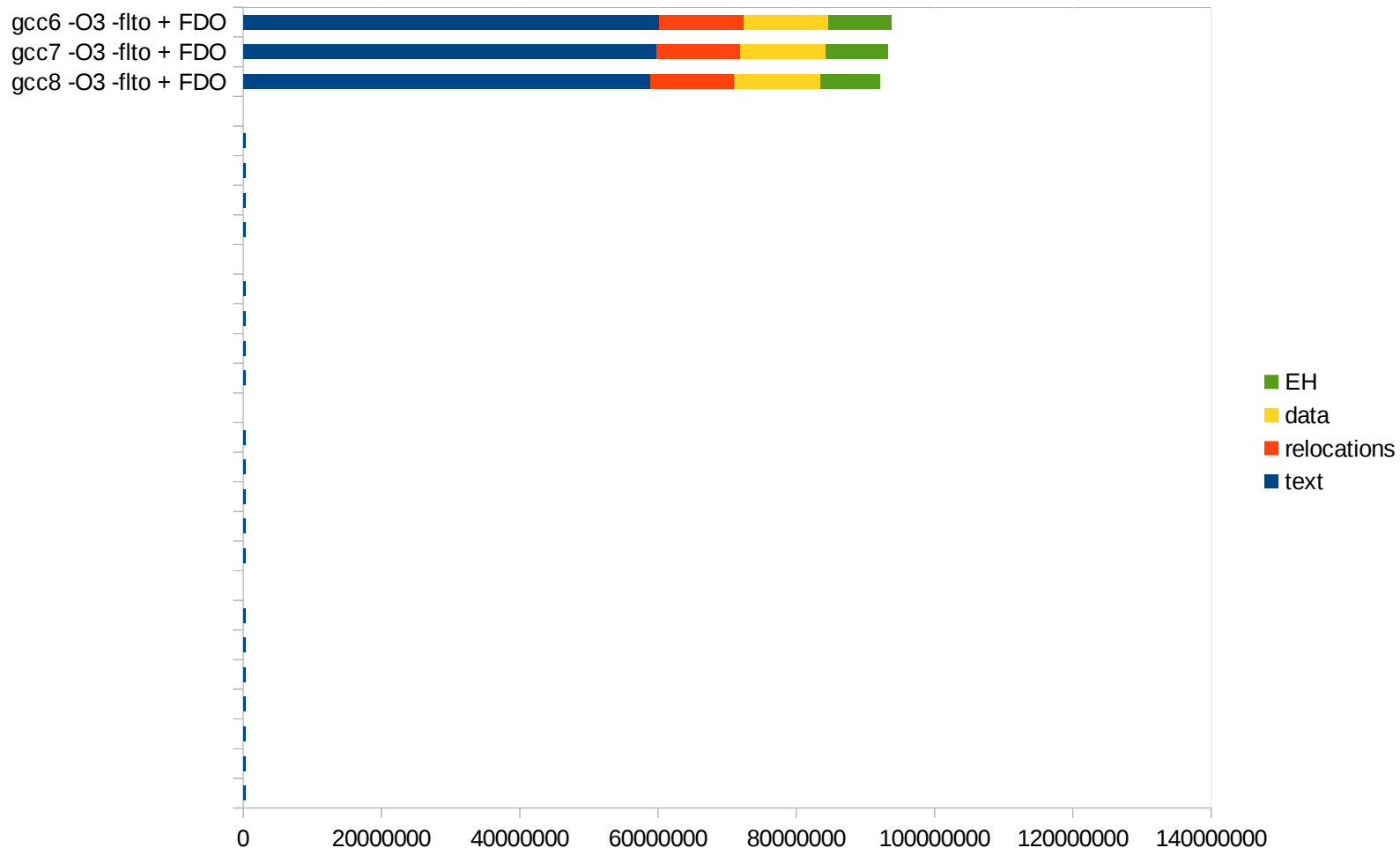
Firefox performance – dromaeo DOM



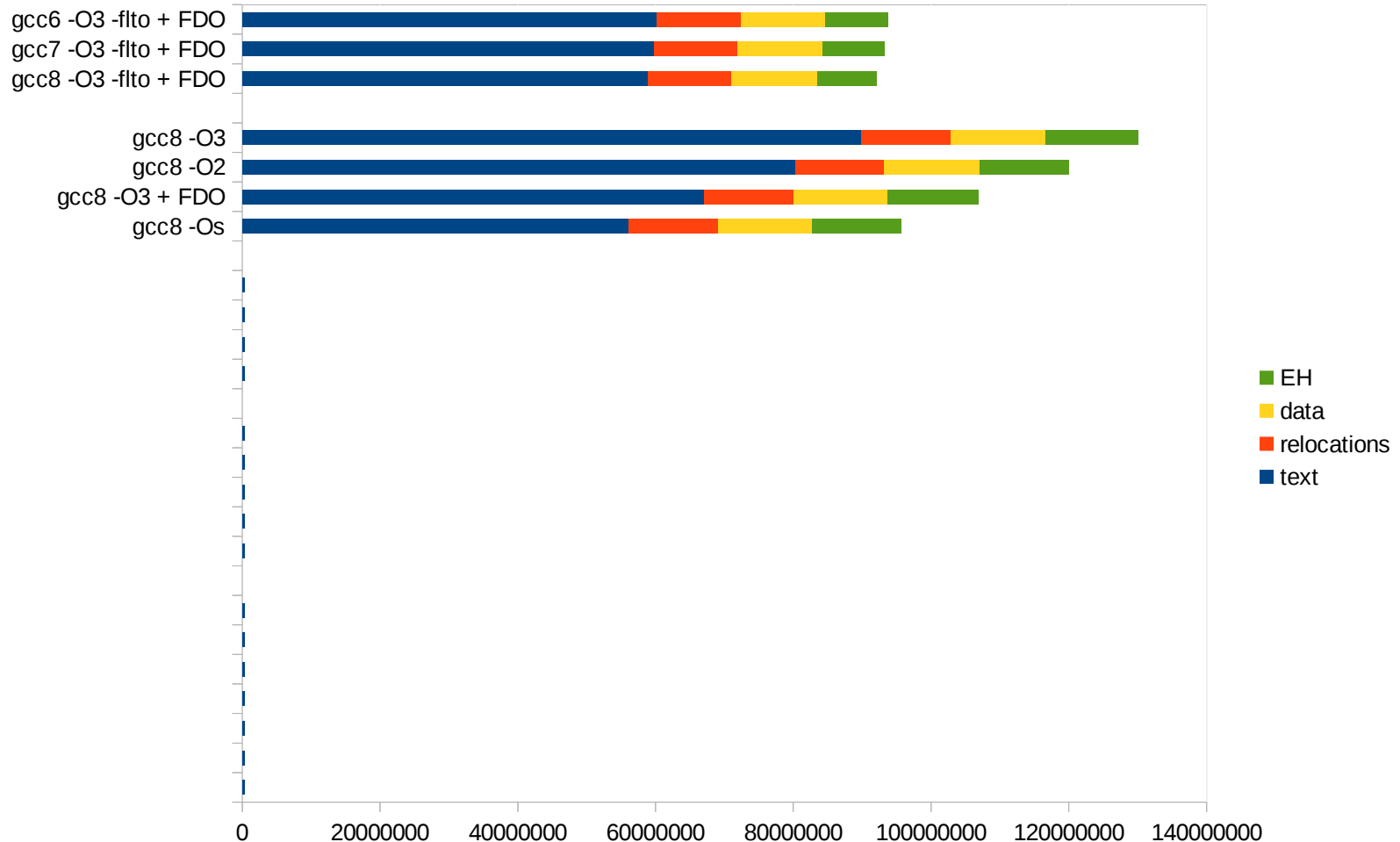
Firefox performance – tp50 page responsiveness



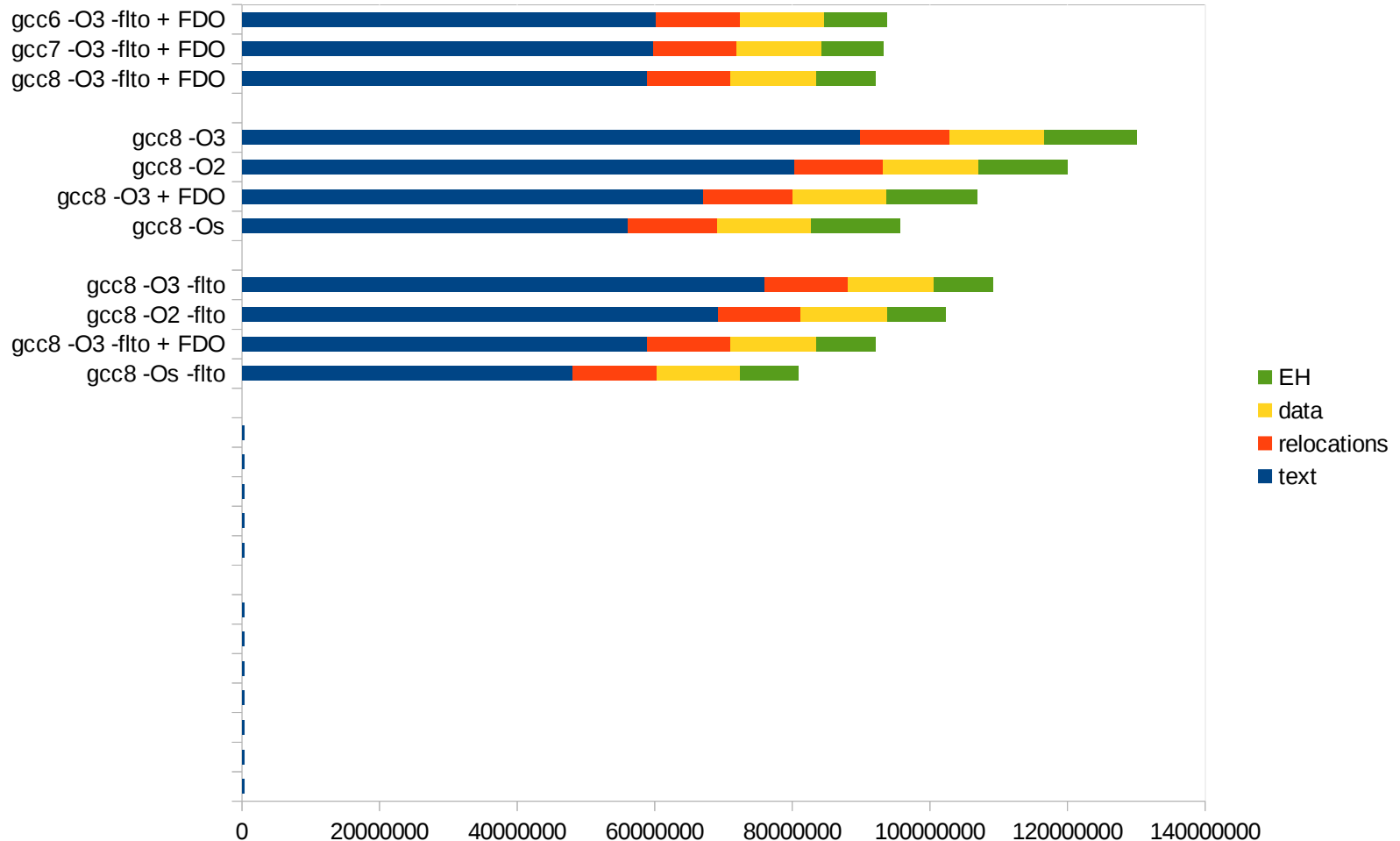
Firefox binary size



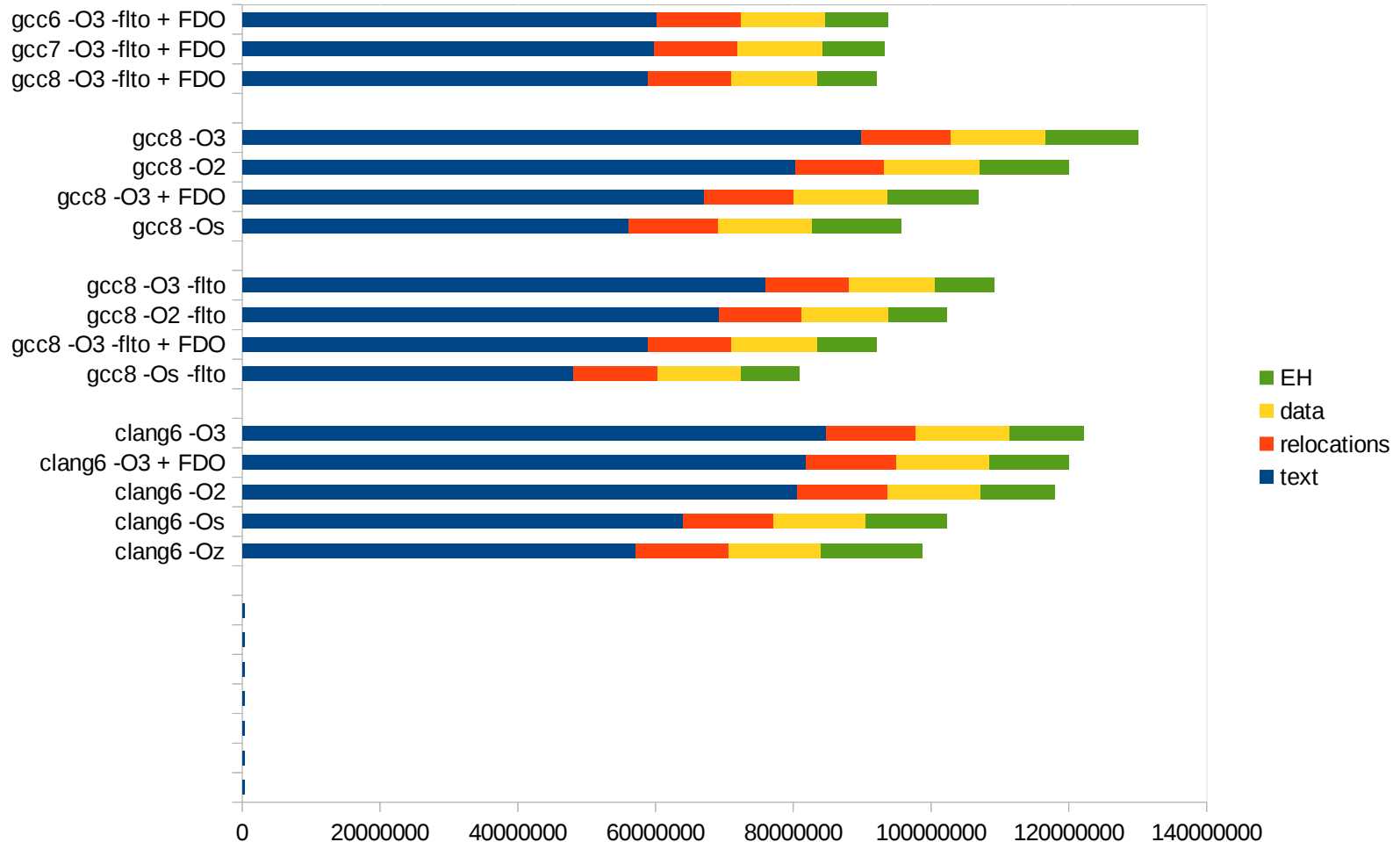
Firefox binary size



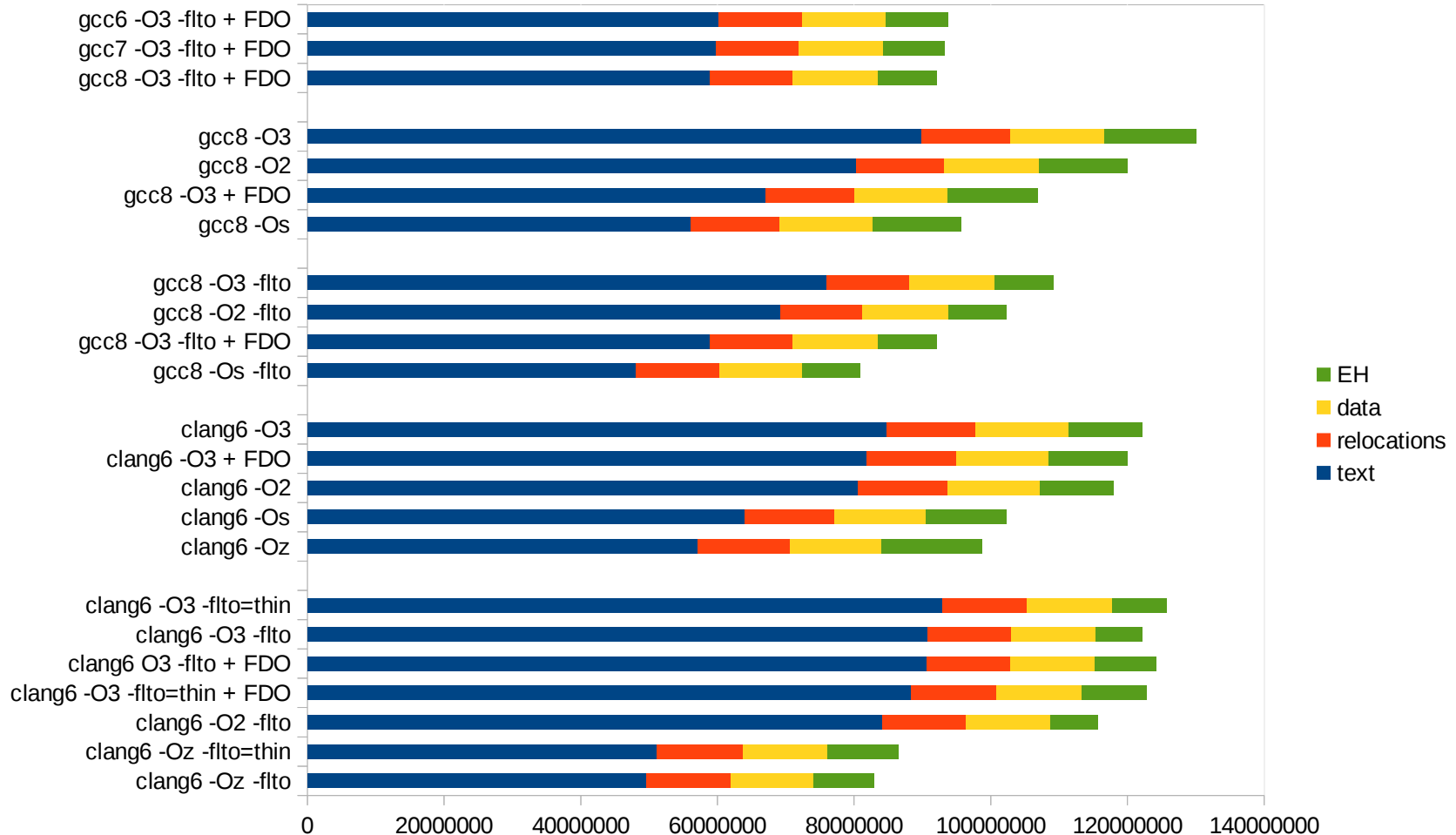
Firefox binary size



Firefox binary size



Firefox binary size



Summary

- LTO now works and scale for large applications (Firefox, Libreoffice,...)
- Important size optimization especially for large C++ programs
- Often important performance optimization especially when combined with profile feedback
- Space for future improvements (both on GCC side and re-optimization of programs to LTO model)



LTO in openSUSE Factory

The background features a large teal shape on the left and a green shape on the right, separated by a white diagonal line. The teal shape has a white arrow-like cutout on its right side that points towards the green shape.

LTO in openSUSE Factory

- A new staging project (openSUSE:Factory:Staging:N)
- Setting: *Optflags: *-flt0*
- ~80 packages failed (out of all ~2300)
- Branch all failing packages and disable LTO
- LTO Factory (w/ some disabled packages) can run openQA test-suite (with a small fallout)



Package statistics

- Total ELF files in distribution: ~6700
- Reduction: 1839 MB to 1736MB (-5.6%)
- libmergedlo.so: reduction 10MB (-16%)
- Biggest reduction:
 - mysql_upgrade (-92%)
 - Innochecksum (-94%)
 - mbstream (-94%)



Issues

- Argyllcms: lto1: fatal error: multiple prevailing defs for 'xcal_read_icc':
LD PR: <https://sourceware.org/PR23079>
- GCC LTO miscompilation: <http://gcc.gnu.org/PR85248>
- *.symver* in shared libraries: <https://gcc.gnu.org/PR48200>:
 - `__asm__(".symver old_foo,foo@VERS_1.1")`
 - New *symver* function attribute must be added (GCC 9.1.0)



Issues (cont.)

- static libraries (*.la):
 - e2fsprogs, btrfsprogs, ...
 - Fat LTO objects must be used (-ffat-lto-objects)
 - LTO elf sections should be stripped
 - OBS sanitizer should be extended
 - LTO mode can combine both LTO objects and assembly objects



Issues (cont. 2)

- LTO warnings (-Wodr, ...):
 - *ltrace*: error: type of 'filter_matches_symbol' does not match original declaration [-Werror=lto-type-mismatch]
 - *gdb*: error: type 'struct ipa_sym_addresses' violates the C++ One Definition Rule [-Werror=odr]
- mpir: weak configure script that scans *.o file as a blob
- weak support for LTO debug info in *dwz* tool
- Maybe higher memory constraints for selected packages



Issues (cont. 3)

- Usage of top-level assembler (<https://gcc.gnu.org/PR57703>):
 - Example of *syscall.cc* in Chromium project:

```
asm volatile(".text\n"
```

```
    ".align 16, 0x90\n"
```

```
    ".type SyscallAsm, @function\n"
```

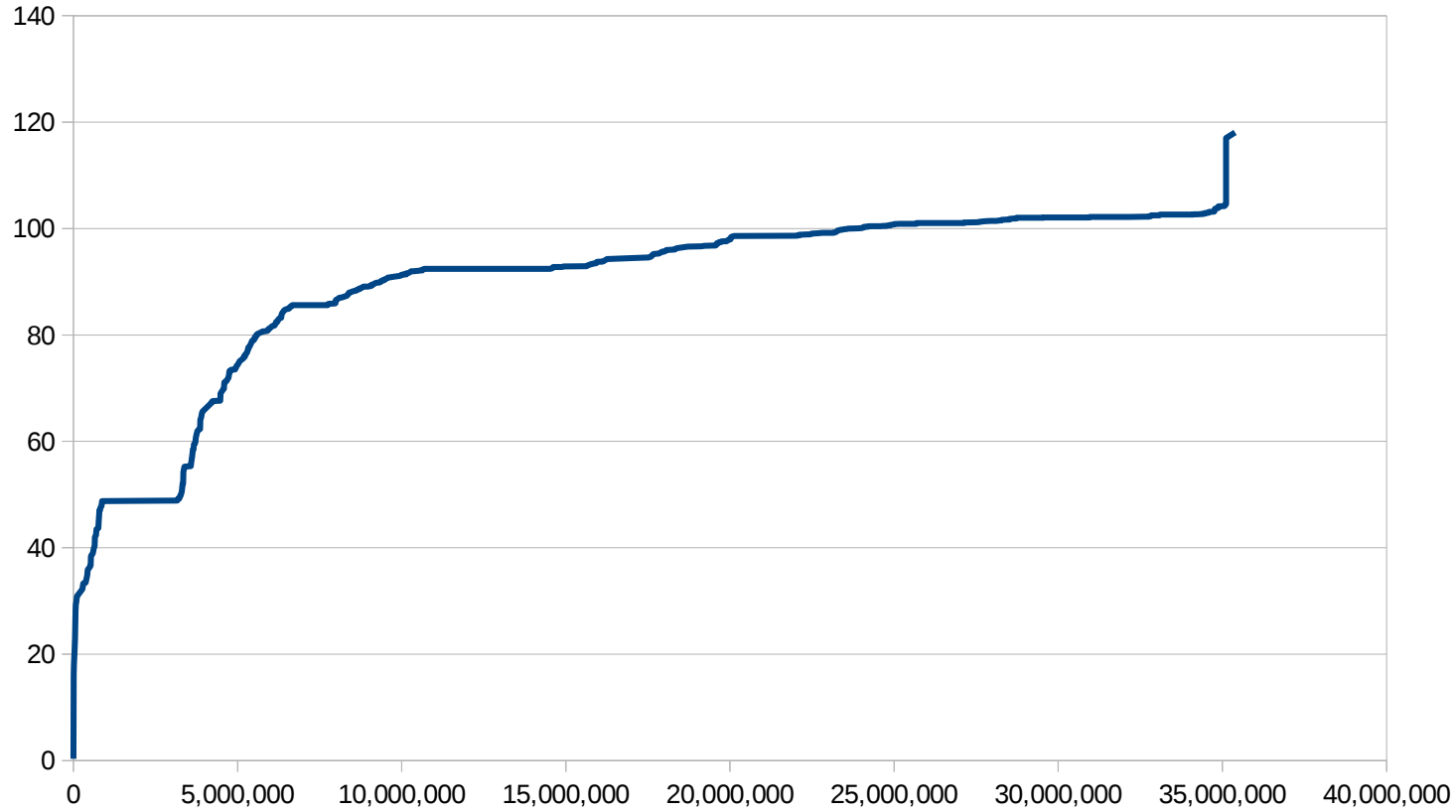
```
"SyscallAsm:.cfi_startproc\n"
```

- Error:*nacl_helper.ltrans1.ltrans.o: In function `playground2::SandboxSyscall(int, long, long, long, long, long)`:*

```
nacl_helper.ltrans1.o:(.text+0x4503): undefined reference to `SyscallAsm'
```



Histogram of text segment sizes



Conclusion

- LTO looks mature enough to be used by default
- Do we want it for openSUSE Factory?
- For the future, can we offer it also for SLE?



License

This slide deck is licensed under the Creative Commons Attribution-ShareAlike 4.0 International license. It can be shared and adapted for any purpose (even commercially) as long as Attribution is given and any derivative work is distributed under the same license.

Details can be found at <https://creativecommons.org/licenses/by-sa/4.0/>

General Disclaimer

This document is not to be construed as a promise by any participating organisation to develop, deliver, or market a product. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. openSUSE makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. The development, release, and timing of features or functionality described for openSUSE products remains at the sole discretion of openSUSE. Further, openSUSE reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All openSUSE marks referenced in this presentation are trademarks or registered trademarks of SUSE LLC, in the United States and other countries. All third-party trademarks are the property of their respective owners.

Credits

Template

Richard Brown
rbrown@opensuse.org

Design & Inspiration

openSUSE Design Team
<http://opensuse.github.io/branding-guidelines/>