Clang vs. GCC for SPEC2017 on AArch64

Sjoerd Meijer

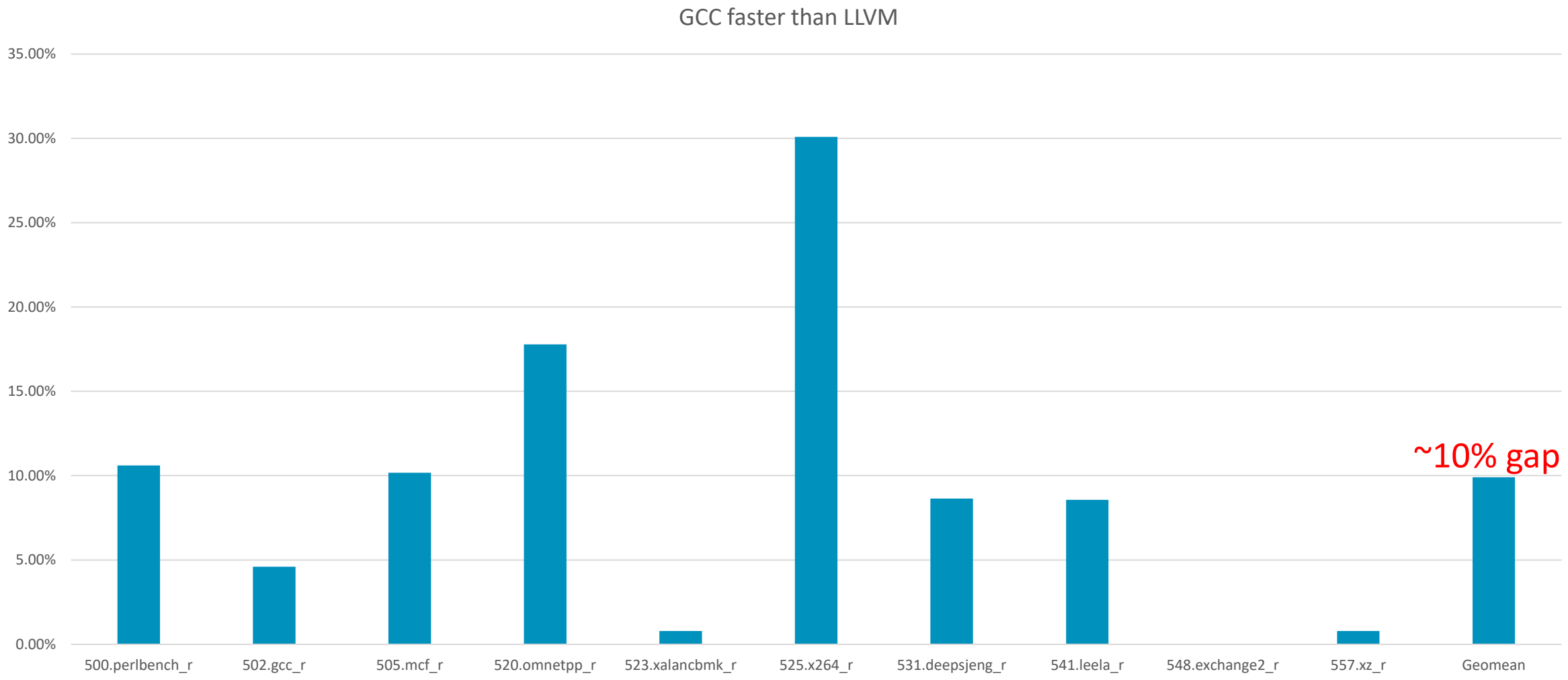
Sjoerd.meijer@arm.com

US LLVM Developer conference 2021

SPEC2017

- "performance measurements that can be used to compare compute-intensive workloads on different computer systems."
- Family of benchmarks: SPEC2006, SPEC2017.
  - Sub-suites for measuring integer and floating-point performance.
  - We will focus on SPEC2017Int rate.
- Contribute generic improvements, I.e. not only improve SPEC.
  - Target AArch64, but
  - will present work on target independent optimisations.

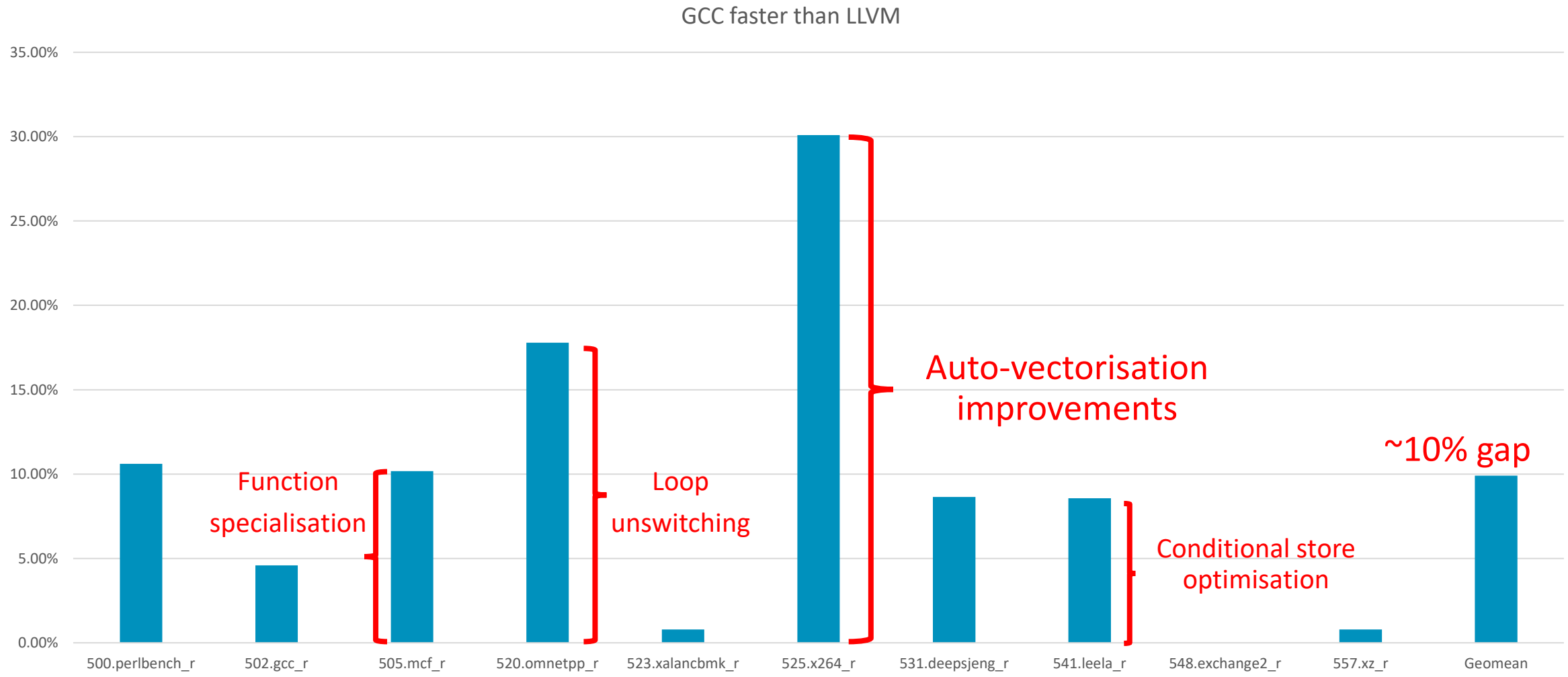
LLVM12 vs. GCC11 –Ofast



* Graphs courtesy of Sanne Wouda, Kyrylo Tkachov

arm

LLVM12 vs. GCC11 –Ofast



arm

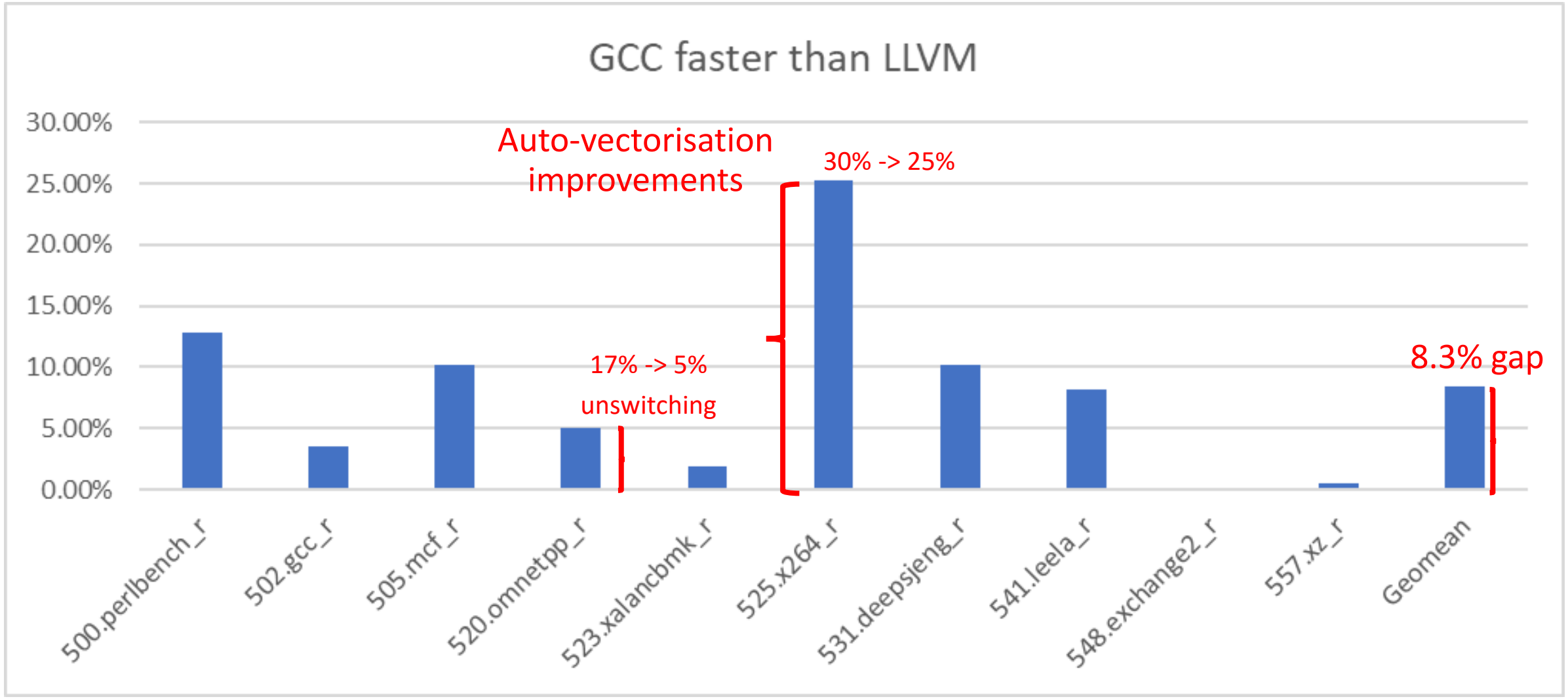
Fixed

- D99354: Loopunswitching (omnetpp), JinGu.
  - Hoist loop-invariant control flow.
  - Port partially invariant unswitching from LoopUnswitch to SimpleLoopUnswitch.
  - Allows elimination of a dead loop.

- D107281: Added a new pass function specialisation (MCF), Sjoerd.
  - Not yet enabled by default.

- Smaller auto-vec improvements (x264)
  - Added a match pattern, Dave
  - Loads of i8 vectors, and cost-modeling, Sjoerd

LLVM12 vs. GCC11 –Ofast, update



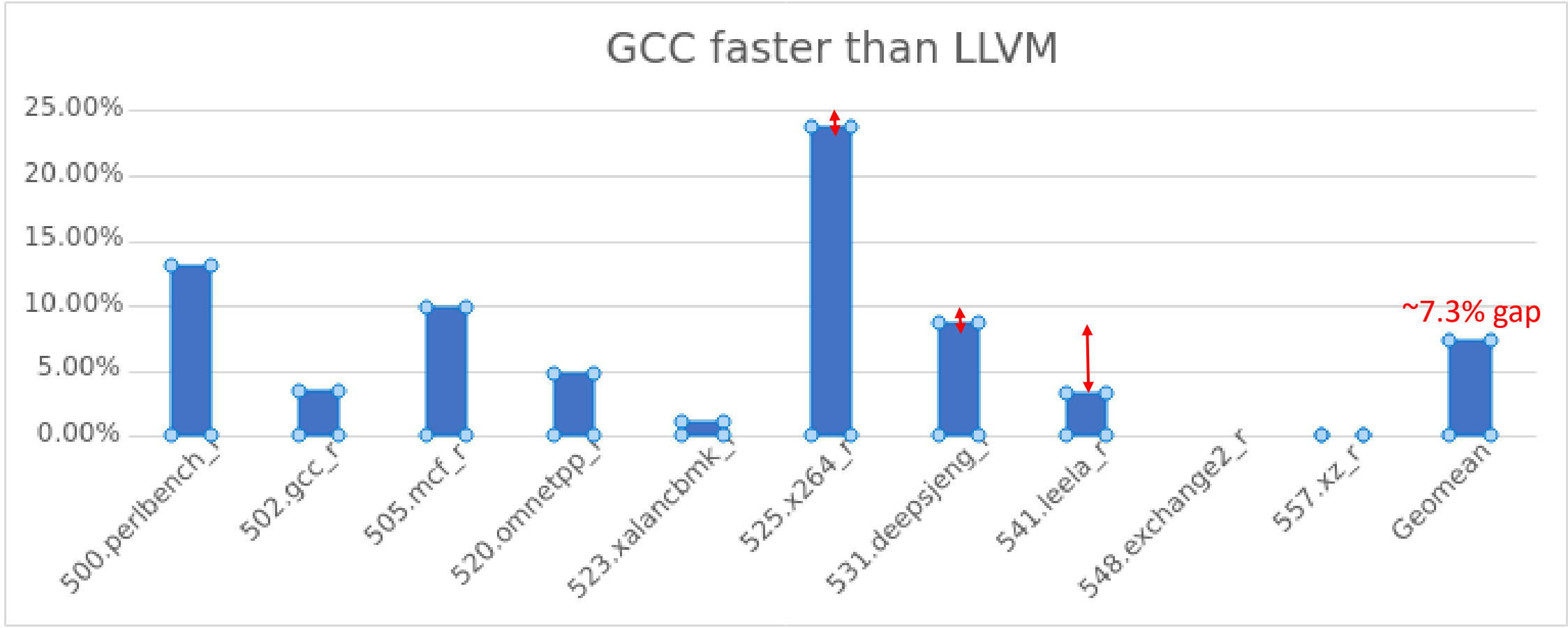
arm

Fixed, cont'd

- D107281: SimpifyCFG (leela), Momchil.
  - Speculate a store preceded by a local non-escaping load.

- Loopunrolling (perlbench), JinGu:
  - D107381: Extends runtime unrolling for loops with multiple exiting blocks.
  - D105996: Enable unrolling of non-constant bounds for all AArch64 targets.

LLVM12 vs. GCC11 –Ofast, update #2



arm

Observations

- Target independent IR passes and optimisations
  - I.e., nothing AArch64 specific.

- Identified passes that:
  - Heavily interact with each other, e.g. unroller and the vectorisers.
  - Are missing functionality, e.g. the SLP vectoriser.
  - Are not enabled by default, e.g. GVNHoist, function specialisation.
  - Are not generic enough, e.g. loop-distribute.

Pass Ordering Challenges

- X264 has kernels with small, constant integer bounds:
  - Fully unrolled first -> Loop vectoriser doesn't run -> SLP vectoriser is missing features.

```
void fn( int16_t *A, uint16_t *B, uint16_t *C )
{
   for( int i = 0; i < 16; i++ ) {
      if (A[i] > 0 )
         A[i] = (C[i] + A[i]) * B[i] >> 16;
       else
          A[i] = - ((C[i] - A[i]) * B[i] >> 16);
   }
}
```

- D102748: Don't unroll before vectorisation.
  - PR47178, PR47726, PR47554, PR47436, PR31572, PR47553, PR47491.
  - Consensus:
    - full unrolling happens early: scalar optimizations have a chance to work on fully unrolled loops
    - SLP vectorizer needs fixing.

Passes Lacking Features

- [SLPVectorizer] Implement initial memory versioning: D102834
  - Emit runtime alias checks, like the loop vectoriser.

```
void fn( int16_t *A, uint16_t *B, uint16_t *C )
{
   for( int i = 0; i < 16; i++ ) {
      if (A[i] > 0 )
         A[i] = (C[i] + A[i]) * B[i] >> 16;
       else
          A[i] = - ((C[i] - A[i]) * B[i] >> 16);
  }
}
```

- Other, target dependent optimisations:
  - If-conversion
  - Cost-modeling

Passes Not Enabled

- Function Specialisation:
  - TODO: Is it generic enough, is worth the extra compile-times?

- GVNHoist:
  - Enabled/reverted a few times, still correctness and perf problems
  - [GVN] Simple GVN hoist – scalars: D110817
  - [GVN] Simple GVN hoist - loads and stores: D110822

Passes Not Generic Enough

- Loop-distribute:
  - Helps hmmer in SPEC2006.
  - Improve loop distribute cost model: D100381
  - But were not able to make this generic.

Pass Missing?

- Loop Invariant Code Motion (LICM)
  - Philosophy: canonicalisation transformation.
  - Aggressively hoists everything it can, backend should undo this.
  - Except, we don't have a good solution:
    - LoopSink: works on IR and only with profile info
    - MachineSink: can't sink into loops.

Various other TODOs

- Missed opportunities for register promotion: PR51193
- Multiple evaluations of a GEP: PR51184
- Sub-optimal placement of loop exit blocks: PR51185
- Failure to recognize table-based ctz implementation: PR46434

Conclusion

- Have some way to go:
  - Auto-vectorisation (SLP vectoriser)
  - Hoisting
  - Undoing LICM?
  - Rinse & Repeat the whole process

- Feedback welcome!
  - Please get in touch on the dev list, phabricator, or email.

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
תודה