Examining and mitigating the performance impact of C++ exceptions on the non-exceptional path in LLVM

Presenter:

Modi Mo

Meta

Background

How C++ exceptions are modeled in LLVM

Hampering Inlining

```
lpad:                                             ; preds = %entry
; *cost* *= 5*
  %1 = landingpad { i8*, i32 }
           cleanup
; *cost* *= 35*
  call void @_ZN1AD1Ev(%struct.A* nonnull dereferenceable(1) %Cleanup) #5
; *cost* *= 5*
  resume { i8*, i32 } %1
```

- Simple landing pads cost 45 inline budget
- -O3 has a default max budget of 250
- More complex landing pads, such as destroying a vector of objects, cost 125

More Conservative Memory Aliasing

Capture Tracking, used in general Alias Analysis has a limit on value usage

- llvm-project/llvm/lib/Analysis/CaptureTracking.cpp

```
static cl::opt<unsigned>
DefaultMaxUsesToExplore("capture-tracking-max-uses-to-explore", cl::Hidden,
                        cl::desc("Maximal number of uses to explore."),
                        cl::init(20));
```

- If a value is used >20 times the pointer is considered captured
- Landing pads introduce additional uses

Building Clang -fexcept vs -fno-except

- 34% more captured pointers → 7% fewer instructions hoisted by LICM

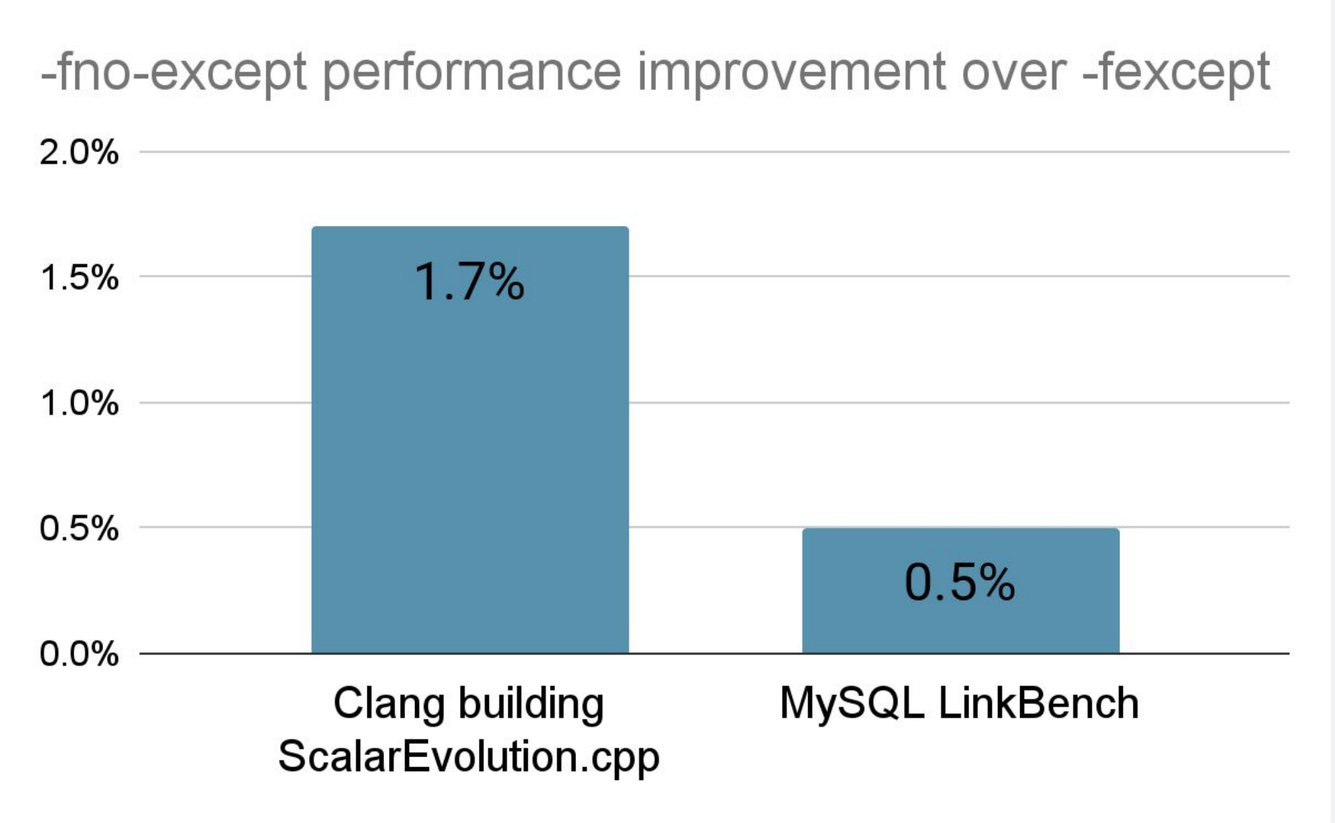
Sub-optimal Register Allocation

```
ClassWithDestructor Object;

for (const auto& iter : iterable) {
    MayThrow1();
}
...
MayThrow2();
{
    ...
    MayThrow3();
}
```

- Each call has a CFG edge to the landing pad to destroy Object
- If RA places Object in non-volatile register or stack in the landing pad, it must be there before **every** call
- Lowers RA flexibility

Performance -fno-except vs -fexcept



https://github.com/facebookarchive/linkbench

Mitigating the costs:
Lowering the number of landing pads needed

Removing largest propagator of exceptions

The largest bottom-up propagator of exceptions is “operator new”.

However, not many programs can or expect to recover from OOM but still pay the costs of landing pads from the possibility of exceptions.

New Clang FE flag: **-fnew-infallible** (https://reviews.llvm.org/D105225)

- Expresses that “operator new” cannot fail and terminates if an exception occurs
- Results in a 9.3% increase in functions without exceptions in Clang ThinLTO build

Improve NoUnwind attribute propagation

Calls outside of a TU, unless annotated, have to be considered potentially throwing.

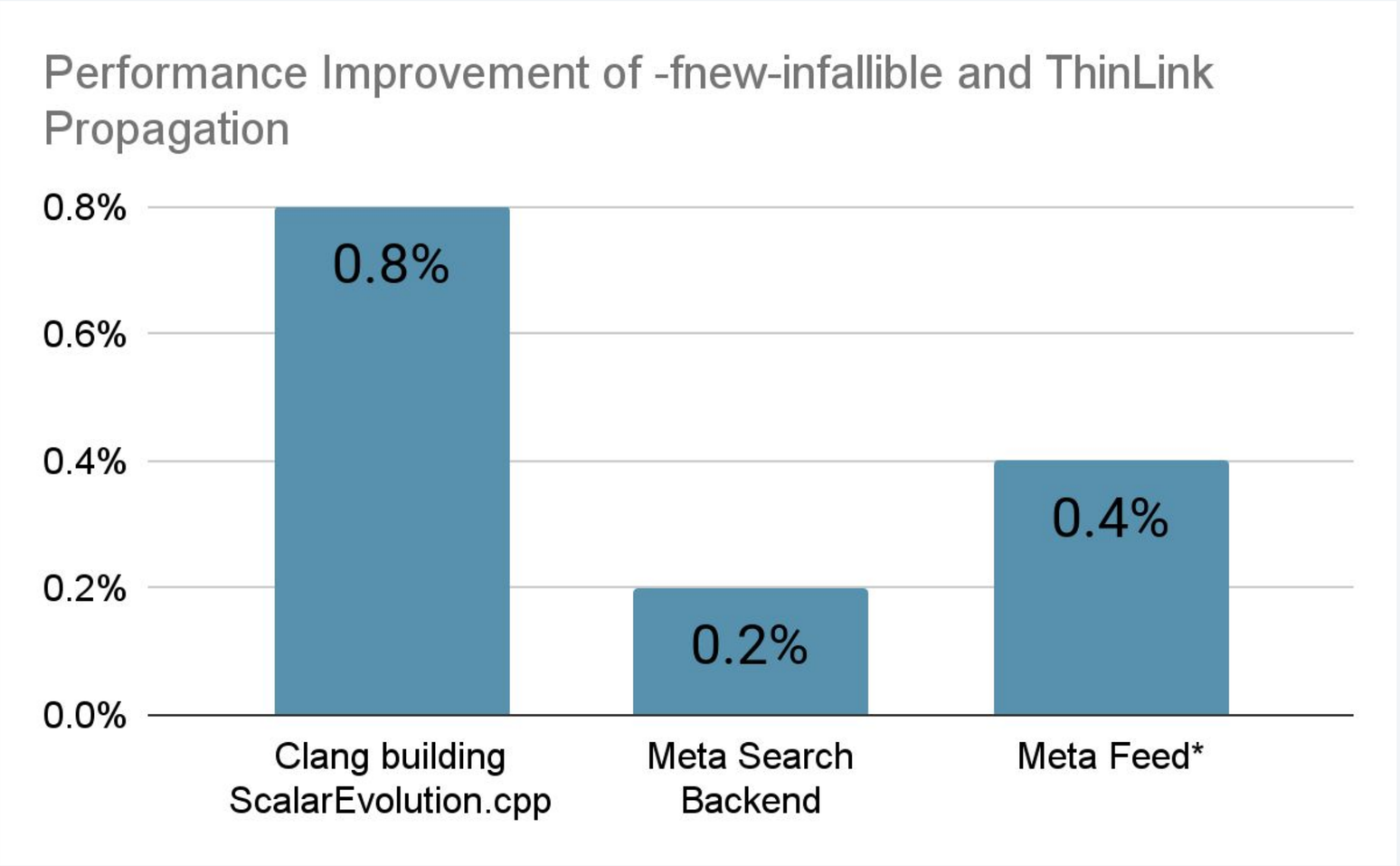
Propagating whether these functions actually throw across TUs in ThinLTO helps eliminate this pessimistic analysis.

Propagating attributes during ThinLink (https://reviews.llvm.org/D36850)

- ThinLTO summaries are used to propagate NoUnwind across the call-graph of the entire program
- -Wl,-mllvm,-disable-thinlto-funcattrs=0
- Results in a 9.9% increase in functions without exceptions in Clang ThinLTO build

Results:

-fnew-infallible and ThinLink Propagation



*Not conclusive due to measurement difficulties

Future Directions

Lowering the number of landing pads needed

- Additional exception propagators could be annotated, notably those in libstdc++*.

Lowering costs of having landing pads

- Exception-only code can be outlined early into cold code
  - No longer impacts the inline cost model

*https://gcc.gnu.org/pipermail/gcc/2021-August/236984.html

Questions/Comments?

Leave them on the video and/or find me at the conference!