Using and Improving Optimization Remarks

*Remark*able compiler insights for performance tuning

Tobias Stadler
mail@stadler-tobias.de
in collaboration with Florian Hahn, Jon Roelofs, Arnold Schwaighofer, Gerolf Hoflehner @ Apple

LLVM Developers' Meeting '25, Santa Clara, California
2025-10-28

Motivation

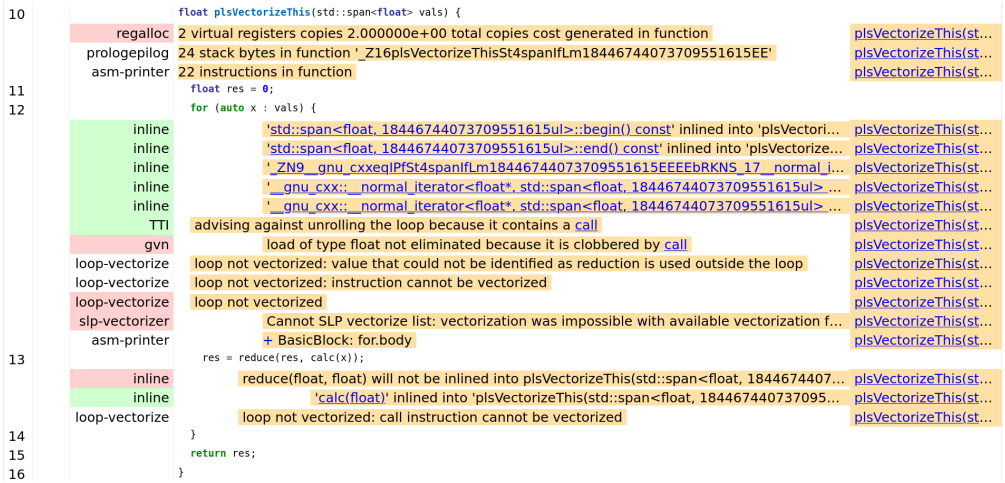
```
float plsVectorizeThis(std::span<float> vals) {
  float res = 0;
  for(auto x : vals) {
    // e.g. reduce: a + b; calc: x * x;
    res = reduce(res, calc(x));
  }
  return res;
}
```

```
foo.cpp:9:12: remark: loop not vectorized: cannot prove it is safe to reorder floating-point
     operations; allow reordering by specifying '#pragma clang loop vectorize(enable)' before
     the loop or by providing the compiler option '-ffast-math' [-Rpass-analysis=loop-
     vectorize]
    9 |   return a + b;
      |          ^
foo.cpp:13:11: remark: loop not vectorized: call instruction cannot be vectorized
     [-Rpass-analysis=loop-vectorize]
   13 |     res = reduce(res, calc(x));
```

Actionability? Discoverability?

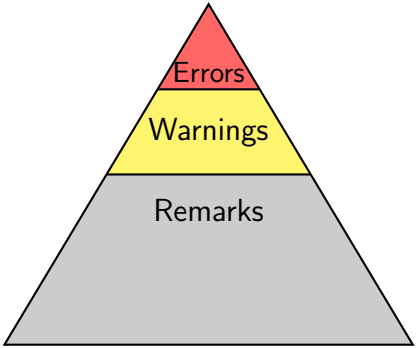
opt-viewer Example

```
python llvm/tools/opt-viewer/opt-viewer.py <yaml files>
```



Digestability? Scalability?

Remarks in the Diagnostic Hierarchy



- Intention: 3rd diagnostic class
- Optimizer, Backend

- `Passed`: Applied optimizations
- `Missed`: Untaken optimization opportunities (e.g. unsafe)
- `Analysis`: Heuristics, statistics, ...

User Groups

Compiler Users

- Understand missed optimizations
- Guide the compiler
- Communicate program invariants
- Override heuristics

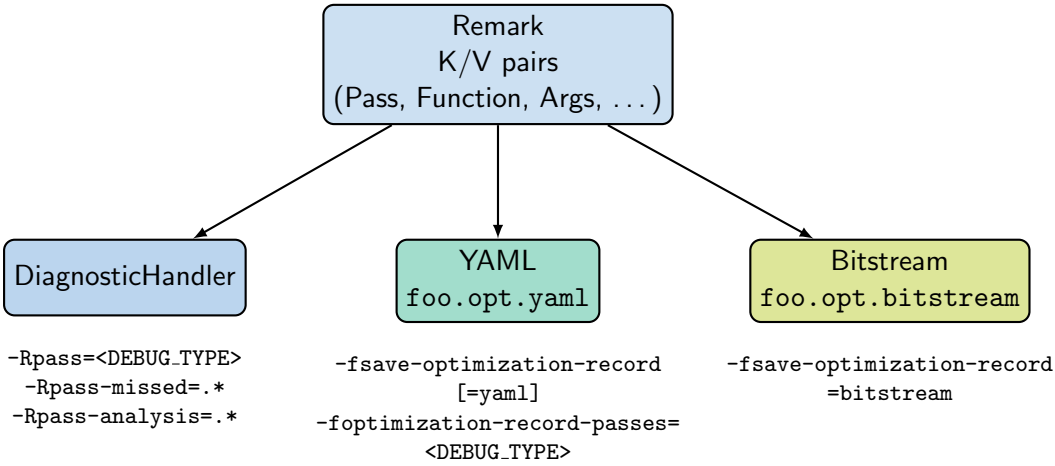
Remarks as teaching tool:
user intent $\leftrightarrow$ compiler capabilities

Compiler Developers

- Debug optimization passes
- Tune heuristics
- Understand regressions
- Estimate blast radius

Remarks as telemetry:
local development & CI tracking

Remark Formats



Remark Landscape

Generation → **Processing** → **Displaying**

- Clang
- MLIR
- Swift

- Inlining
- Vectorization
- Unrolling
- ...

- libRemarks
- llvm-remarkutil
- dsymutil (MachO)
- opt-stats.py
- opt-diff.py

- Diagnostics
- opt-viewer.py
- llvm-opt-report
- Compiler Explorer

Fragmentation?

Bitstream Remarks

Motivation: Scalability

- YAML does not scale (entire code bases, collection in CI, ...)

Issues

- Supported on MachO only
- String table in object file
- Incompatible with `opt`

Overhauling the Bitstream Format

- Bitstream files now fully standalone (Drop-in YAML replacement!)
- Format auto-detection
- More efficient encoding: 40% file size reduction
- Embed blobs: e.g. LLVM BitCode

Regarding removal of `SerializerMode`, see docs

```
Expected<LLVMRemarkFileHandle> setupLLVMOptimizationRemarks(...);
```

llvm-remarkutil filter: The multi-tool for bulk processing

```
llvm-remarkutil filter --remark-type=missed --pass-name=inline -o missed.yaml
    foo.opt.bitstream bar.opt.bitstream
```

- Filter by function, pass, remark name, type, arguments
- Regex support (`--rfunction`, `--rpass-name`, ...)
- Automatic format conversion (YAML $\leftrightarrow$ Bitstream)
- Merge multiple remark files
- `--exclude`
- `--dedupe`, `--sort`

llvm-remarkutil summary: Summarizing Remarks

```
llvm-remarkutil summary -o summary.yaml main.opt.bitstream
```

- Summarize multiple remarks into new remarks using different strategies
- Current strategies:
  - `--inline-callees`: Per-callee inlining statistics
  - Contributions welcome!
- Input remark retention: `--keep=none|used|all`

```
template <class, class _Cp, bool _IsConst, class _Tp, class _Proj, __enable_if_t<__is_identity<_Proj>::value, int> = 0>
_LIBCPP_HIDE_FROM_ABI _LIBCPP_CONSTEXPR_SINCE_CXX20 __iter_diff_t<__bit_iterator<_Cp, _IsConst> >
__count(__bit_iterator<_Cp, _IsConst> __first, __bit_iterator<_Cp, _IsConst> __last, const _Tp& __value, _Proj&) {
- Incoming Calls (Inlined: 1, TooCostly: 2)
Least profitable (cost=255, threshold=250)
Most profitable (cost=30, threshold=250)
```

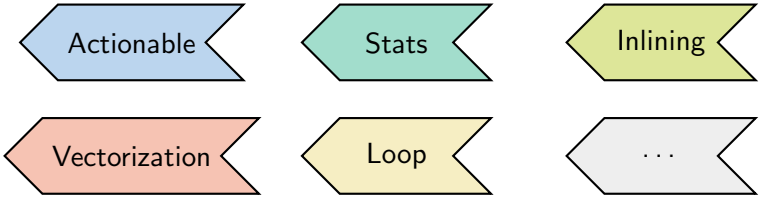
{Function,Pass-}Local ADT Statistics

- Fine-grained `STATISTIC` implementation (using `thread_local`, pass instrumentation)

```
clang++ -fsave-optimization-record -mllvm --stats -O3 main.cpp

--- !Analysis
Pass: sroa
Name: PassStatistics
Function: _ZNSt6bitsetILm64EEC2Ey
Args:
  - MaxPartitionsPerAlloca: '1'
  - MaxUsesPerAllocaPartition: '2'
  - NumAllocaPartitionUses: '4'
  - NumAllocaPartitions: '2'
  - NumAllocasAnalyzed: '2'
  - NumDeleted: '4'
  - NumPromoted: '2'
...
```

Remark Tags: Different Needs, Different Remarks



- Multiple tags per Remark
- Extensible
- Orthogonal to remark type
- Obviate the need for sub-types: `AnalysisFPCommute`, `AnalysisAliasing`

What's next?

More upstreaming tbd:
`github.com/tobias-stadler/llvm-project/tree/remarks-future`

- Expose tags to `OptimizationRemarkEmitter`, start tagging remarks
- Warning-like diagnostic categories based on tags (`-Rvectorize`, ...)
- **Tag-aware `diff` tool** (e.g. subtract Stats)
- Python bindings for libRemarks

Summary

- Remarks close the feedback loop between compiler and developer
- Challenge: How to extract the specific information we need?
- In need of better infrastructure, tooling, and more actionable remarks

- Bitstream format now as drop-in replacement for YAML
- New tools: `llvm-remarkutil filter`, `summary`
- Per-function/pass `STATISTIC`s
- Future: Remark tags for improved organization, filtering, diffing,...
- Contributions welcome!

Thanks!