GSoC 2022 - Remove undef: Move Uninitialized Memory to Poison Final Report

John McIver (mentee)
mciverj@unm.edu

Nuno Lopes (mentor)
nuno.lopes@tecnico.ulisboa.pt

September 12, 2022

Abstract

Current memory semantics of the LLVM-IR use `undef` when dereferencing an uninitialized memory location. This results in `phi` nodes with `undef` operands (instead of `poison`), which prevents further optimizations, such as the folding of `phi` nodes. This Google Summer of Code (GSoC) project implements methodologies for bit-field loading and storing and replacing `undef` returned by uninitialized memory loads with `freeze poison` in SROA, mem2reg, and GVN. This work increases the use of `poison` semantics, contributing to the eventual deprecation of `undef`.

1 Introduction

During the period of performance the following topics developed for community review and patch submission:

- Refactor clang IR generation of C and C++ bit-fields to use `poison` memory semantics
- Refactor memory based optimizations to use `phi` with `poison` rather than `phi` with `undef`

The refactoring of memory based optimizations was expanded from the original proposal beyond scalar replacement of aggregates (SROA) to include mem2reg and global value numbering (GVN), NewGVN, and GlobalOpt. Additionally, refactoring of memory state built-ins used by several optimizations has been achieved.

Prior to the beginning of GSoC I had limited experience with the LLVM framework and patch submission process [4, 3]. During the implementation of this project I gained experience with:

- LLVM optimization framework
- LLVM IR
  - Semantics of `undef`, `freeze` `poison`, etc.
  - Attribute implementation and adaptation
  - Memory access semantics
  - Data structure querying and manipulation
- LLVM regression and test-suite frameworks
- Performance testing and root cause analysis techniques
- Automatic test case reduction

Gaining experience with these topics has greatly improved my understanding of the project's architecture and my ability to identify and develop solutions within the LLVM framework.

2 Bit-field Migration to Poison

A request for comment (RFC) was developed and published on the LLVM Discourse site [6], which outlines three different solutions: emit freeze on all bit-field loads, zero initialize and forego freeze on stack-allocated bit-fields, and freeze based on access analysis. While the proposal was undergoing community review a patch was developed using the emission of freeze on all bit-field loads [2].

Bootstrap builds of the test-suite showed that the patch [2] resulted in as much a 15% increase in run-time for `SIBsim4`. Analysis revealed that the addition of `freeze` instructions in bit-fields resulted in extra `mov` instructions. Refactoring code generation to remove the extraneous `mov` instructions was attempted, however due to time constraints an issue was filed [7].

Two issues discovered during the RFC process. First, an aliasing scheme would need to be introduced in order to allow bit-field freezes based on access analysis. Second, inconsistent use of option `-ffine-grained-bitfield-accesses` can cause incorrect behavior when linking with inter-procedural optimization (IPO). To reduce the risk of IPO fine-grained bit-field inlining interactions, two patches were developed to add an IR function attribute `find_grained_bitfields` [5, 1]. Patch [5] was rejected as it would have introduced language-specific knowledge at the IR level.

3 Load with Automatic Freeze

Current `undef` usage appears around dead operations where loads of uninitialized memory occur. A preliminary RFC was posted to LLVM Discourse outlining three potential solutions [8]:

	Can duplicate?	Can hoist?
1. Load with automatic freeze	No	Yes
2. Load without automatic freeze	Yes	Yes
3. Load with noundef	Yes	No

Currently Clang generally emits option 3, which can also be downgraded to option 2 if hoisting is required. Option 1 was selected for initial implementation as it provides the following:

- Backwards compatible with current IR semantics
- Transparent support for Clang bit-fields
- IPO inlining support for -fine-grained-bitfield-accesses
- No side effects for code generated with `!noundef`

The first optimizations to be augmented with the automatic `freeze poison` semantics were SROA and mem2reg. The optimization mem2reg is used by SROA to promote `alloca` instructions. Unlike the insertion of `undef` constants the insertion of `freeze poison` instructions requires tracking for potential removal.

GVN adaptation has also been implemented, but with modification to the memory built-ins utility function `getInitialValueOfAllocation`, which now returns `poison` on uninitialized memory. This API is currently undergoing refactoring to provide two methods: is initialized to zero and is uninitialized. Similar to mem2reg, GVN partial redundancy elimination (PRE) requires the tracking of `freeze poison` insertions for later removal due to critical edge incompatibilities.

Support for suppressing `freeze poison` in the presence of attribute `!noundef` has only been implemented in mem2reg and SROA.

Analysis using the test-suite has been performed. With modified mem2reg, SROA, and GVN the average increase in function size is about 0.5%. The largest increase in compilation time was observed for SQlite which increased from 27.18 to 28.02 seconds. The largest execution run-time increases were observed for Lua (12.97 to 13.65 sec) and SIBsim4 (2.10 to 2.59 sec). Analysis of SIBsim4 showed that additional move instructions exist in a critical loop. Independent compilations of SQlite and PHP were also performed to further evaluate size and instruction composition changes, which did not deviate from test-suite results.

4 Future Work

Several patches are currently outstanding due to related technical issues. The mentee has agreed to continue work on the project past the period of performance with support from the mentor. A number of patches are waiting on an RFC submission, which currently reside on development branch [9].

4.1 Load with Automatic Freeze

A formal RFC for Load with Automatic Freeze (see Section 3) is currently being developed. Modification and patch submission will need to occur for mem2reg, SROA, GVN, and associated LTO/IPO optimizations. Analysis will be performed to verify that load duplication is not occurring and if it is mitigation techniques will be developed. If during RFC review major technical issues are discovered then alternative semantics will be developed and implemented as per community input.

Simultaneously to RFC release the first patch updating `load` semantics for emission of `freeze poison` on uninitialized memory in LangRef will be submitted. This patch will then be associated to a patch containing mem2reg and SROA for the new semantics. The next patch in the sequence will be the update to `MemoryBuiltings`, followed by a patch containing GVN changes.

4.2 Analysis of Changes

Further analysis will need to be performed to correct performance regressions observed from the aforementioned patches.

4.3 Load with Automatic Freeze `!noundef` Support

When `load` instructions are accessing uninitialized memory and are decorated with `!noundef` then undefined behavior (UB) can be returned. Work is currently underway for mem2reg and SROA with the addition of several regression tests. Development for GVN will also need to be completed.

5 Conclusion

This project has provided a great opportunity to learn about many of the components of LLVM. This has greatly increased my understanding of IR semantics, compiler frameworks, and general compiler development methodologies. GSoC has provided a great start and I look forward to seeing this project to completion.

References

[1] John McIver. 2022. [codegen] add codegen of ir function attribute fine_grained_bitfields. (July 2022). https://reviews.llvm.org/D129542.

[2] John McIver. 2022. [codegen] make uninitialized lvalue bit-field stores poison compatible. (June 2022). https://reviews.llvm.org/D128501.

[3] John McIver. 2022. [instcombine] optimize and of icmps with power-of-2 and contiguous masks. (May 2022). https://reviews.llvm.org/D125717.

[4] John McIver. 2022. [llvm-objcopy] llvm-strip option –only-keep-debug should suppress default –strip-all. (April 2022). https://reviews.llvm.org/D123798.

[5] John McIver. 2022. [llvm][ipo] add ir function attribute fine_grained_bitfields. (July 2022). https://reviews.llvm.org/D129541.

[6] John McIver. 2022. [rfc] making bit-field codegen poison compatible. (June 2022). https://discourse.llvm.org/t/rfc-making-bit-field-codegen-poison-compatible/63250.

[7] John McIver. 2022. Llvm codegen producing extra move instructions or crashing with freeze #56646. (July 2022). https://github.com/llvm/llvm-project/issues/56646.

[8] John McIver. 2022. Remove undef: move uninitialized memory to poison. (July 2022). https://discourse.llvm.org/t/remove-undef-move-uninitialized-memory-to-poison/61123.

[9] John McIver. 2022. Uninitialized memory to freeze poison branch. (September 2022). https://github.com/jmciver/llvm-project/tree/development/memory-load/undef-to-poison.