Performance Impact of Exploiting Undefined Behavior in C/C++

*Lucian I. Popescu, **Nuno P. Lopes
*Faculty of Automatica and Computer Science, University of Politehnica of Bucharest
** Institute of Superior Technology, University of Lisbon

+ lucian.popescu187@gmail.com, **nuno.lopes@tecnico.ulisboa.pt

1. Introduction
Clang and LLVM use undefined behavior (UB) to issue code optimizations. Currently, there is no study that evaluates the performance impact of this class of optimizations. We fill this gap by presenting some early results in this area. Phoronix Test Suite was used to evaluate the performance of a diverse set of applications, including web servers, compression algorithms, graphical environments, etc. By compiling each application with flags that trigger specific UBs, we gathered various metrics (requests per second, MB/s, FPS, etc) for further analysis.

2. Impact
Early results show that in nearly 90% of the cases the performance impact is insignificant (between -2% and 2%).

3. Experiment Setup
The performance tests were run on a machine with the following specs:
- Processor: 2 x Intel Xeon E5-2690 v2 @ 3.60GHz (20 Cores / 40 Threads), MicroArch: IvyBridge
- OS: Debian 11, kernel: 5.10.0-21-amd64
- Compiler: Clang 15.0.7

The experiments were conducted using the following steps:
- Compile the benchmark with no UB flags enabled (baseline)
- Compile the benchmark using one UB flag at a time
- Run baseline using Phoronix and fetch the results
- Run the benchmarks with UB flags using Phoronix and fetch the results
- Compare the UB flags benchmark results with the baseline

4. Undefined Behavior Flags
Flag name | Flag Description
--- | ---
-fwrapv | Treat signed integer overflow as two’s complement. Drops ‘nsw’ from IR.
-fno-strict-aliasing | Don’t use type-based alias analysis.
-fstrict-enums | Enable optimizations that take advantage of enums’ value ranges. Adds ‘range’ from IR.
-fno-delete-null-pointer-checks | Assume that programs can safely dereference null pointers.
-fno-finite-loops | Don’t assume that all loops are finite. ‘mustprogress’ is not added to any loop or function.
-fconstrain-shift-value | Mask shift RHS so it doesn’t produce poison when RHS >= bitwidth.
-fno-constrain-boolf-value | Drops ‘range’ from IR.
-fno-use-default-alignment | Use alignment of 1 for all memory operations including load, store, memory, etc. Global variables and alloca’s remain unaffected.

5. Future work
- Implement and benchmark other classes of UB.
- Run the benchmarks on different hardware architectures (AMD, ARM).
- Combine the flags when compiling the benchmarks.
- Find a method of discovering new UBs (maybe using Alvey2).
- Run the benchmarks taking into account LTO and PGO.