# The State of PGO in LLVM

## Latest updates:

<https://llvm.org/bugs/show_bug.cgi?id=22718>

<https://llvm.org/bugs/show_bug.cgi?id=22716>

<https://llvm.org/bugs/show_bug.cgi?id=22719>

## Fixes needed

1. Need to faithfully represent the execution count taken from dynamic profiles. Currently, MD\_prof does not really represent an execution count.This makes things like comparing hotness across functions hard or impossible. We need a concept of global hotness.

2. When the CFG or callgraph change, there need to exist an API for incrementally updating/scaling counts. For instance, when a function is inlined or partially inlined, when the CFG is modified, etc. These counts need to be updated incrementally (or perhaps re-computed as a first step into that direction).

## PGO places to gain

* **Inliner**
  + Needs to use profile information and update it accordingly. This is predicated on Chandler's work on the pass manager, of course. Need to represent global profile summary data. For example, for global hotness determination, it is useful to compute additional global summary info, such as a histogram of counts that can be used to determine hotness and working set size estimates for a large percentage of the profiled execution.
* **PRE (particularly of loads!),**
* **the vectorizer** 
  + (i.e. duplicate work down both a hot and cold path when it can be vectorized on the hot path),
* **LoopUnswitch**
* **IRCE, & LoopUnroll** 
  + (avoiding code size explosion in cold code).
  + (Reducing code size can improve performance of course.)
* **Code layout** 
  + **bb layout**
  + **Function Layout**
    - By function layout, I meant layout of functions within the module and then the executable. This could simply be marking/separating hot vs cold functions, or could be fancier via a linker plugin to use profile data to colocate functions with affinity.
  + Function splitting
* **Implementing Profile-Guided Speculative Code Motion in LLVM**
  + ∗<http://users.ece.cmu.edu/~jamiel/15-745/milestone/milestone.pdf>
* **Cache-Friendly Profile Guided Optimization M.Sc. Thesis**
  + <http://baptiste-wicht.com/publication_store/sampling_pgo.pdf>
* **Virtual Call Speculation**
  + Anecdotally evidence says game programmers do not make virtual calls.
  + If a virtual call, or other call through a function pointer, frequently targets a certain function, a profile-guided optimization can insert a conditionally-executed direct call to the frequently-targeted function, and the direct call can be inlined.
* **Register Allocation**
  + Make sure to look at live ranges and then give some preference to keep things in register that live longer.
* **Conditional Branch Optimization**
* With the value probes, profile-guided optimizations can find if a given value in a switch statement is used more often than other values. This value can then be pulled out of the switch statement. The same can be done with if/else instructions where the optimizer can order the if/else so that either the if or else block is placed first depending on which block is more frequently true.
* **Dead Code Separation**
  + Code that is not called during profiling is moved to a special section that is appended to the end of the set of sections. This effectively keeps this section out of the often-used pages.
* **EH Code Separation**
  + The EH code, being exceptionally executed, can often be moved to a separate section when profile-guided optimizations can determine that the exceptions occur only on exceptional conditions.
* **Memory Intrinsics**
  + The expansion of intrinsics can be decided better if it can be determined if an intrinsic is called frequently. An intrinsic can also be optimized based on the block size of moves or copies.

## Making sure we get good data

* Having representative training runs is pre-requisite for using FDO/PGO.
* Tools that compare profiles will be needed to make sure we have good data.

Yes -- there are two aspects of the problems.

1) raw profile data representation in IR and

2) the profile count data represented for CFG.

What you said is for 2) which is one of the possibilities. There is a

third issue that is also going to be covered in more detail -- that is

the Block Frequency propagation algorithm is limited (leading to

information loss). When profile count is available, block frequency

data can be directly computed via simple normalization and scaling.

This requires the raw edge count data to be represented in 1)

truthfully.

## Evidence of PGO/FDO being Beneficial

* <http://llvm.org/devmtg/2013-04/novillo-slides.pdf>

## Ideas worth Exploring

* Use LLDB to set breakpoints and pause and have developer interactively search profile.
  + <http://lldb.llvm.org/tutorial.html>
  + <http://rsquared.sdf.org/gdb/abaw.html>

## Tutorials for figuring out stuff

* How to insert an instruction
  + <http://www.opensource.apple.com/source/lldb/lldb-69/llvm/examples/Fibonacci/fibonacci.cpp>
  + <http://www.opensource.apple.com/source/lldb/lldb-69/llvm/examples/>
* Coverage Mapping
  + <http://llvm.org/docs/doxygen/html/CoverageMapping_8h_source.html>
  + CodeGenPGO
    - <http://clang.llvm.org/doxygen/classclang_1_1CodeGen_1_1CodeGenPGO.html>

## Other Resources

* Gooda
  + <https://github.com/David-Levinthal/gooda>
* Profile Function or Loop Execution Time
  + <https://software.intel.com/sites/products/documentation/doclib/iss/2013/compiler/cpp-lin/GUID-96F454BF-364A-40C9-9B55-BFFAA8FD171D.htm>
* Profile Guided Optimizations
  + <https://msdn.microsoft.com/en-us/library/e7k32f4k.aspx>
* Efficient Profiling in the LLVM Compiler Infrastructure
  + <http://llvm.org/pubs/2010-04-NeustifterProfiling.pdf>
* LLVM Visualization Tool
  + <http://llvm.org/svn/llvm-project/television/trunk/docs/UserGuide.html>
* SamplePGO
  + <http://dl.acm.org/citation.cfm?id=2688366>
* When to use PGO
  + <https://developer.apple.com/library/ios/documentation/DeveloperTools/Conceptual/xcode_profile_guided_optimization/pgo-using/pgo-using.html>



