— IR Around the 🗺 –
Statistical Analysis of a Massive Multi-Language Corpus of IR

Andrew Kallai [1] Khoi Nguyen [2,5] Ludger Paehler [3,5] Aiden Grossman [4,5] Johannes Doerfert [5] Sunita Chandrasekaran [1]

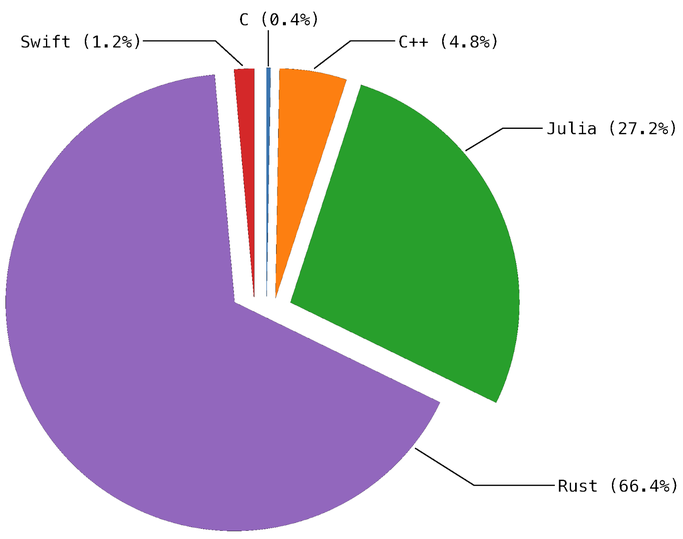
[1]University of Delaware [2]University of California, Berkeley [3]Technical University of Munich [4]University of California, Davis [5]Lawrence Livermore National Laboratory

Abstract

- Statistical analysis demonstrates various relationships between different features of the LLVM's optimization pipeline.
- Outlier extraction toolset provides insights into functions causing runtime abnormalities, giving opportunities for further analysis and optimization.

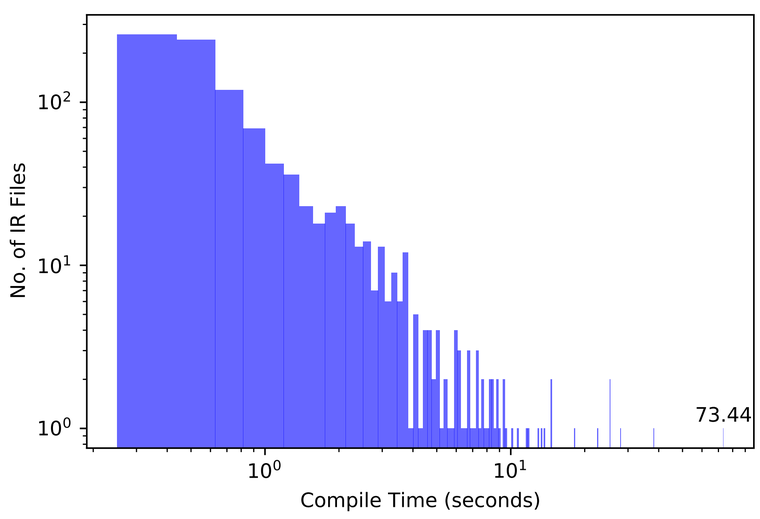
The ComPile Dataset ($n^{\dagger}_{modules} = 402751$)

- A large IR-level dataset from production sources.



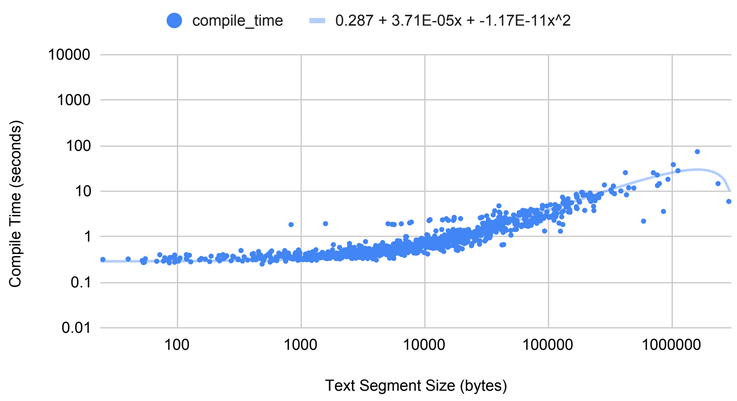
Programming Language	Bitcode *(GB)*	Deduplicated Bitcode *(GB)*	Licensed Bitcode *(GB)*	Licensed Text *(GB)*
C	16	8	2	10
C++	109	74	29	103
Julia	200	184	164	1088
Rust	656	580	400	1524
Swift	8	7	7	36
Total	990	853	602	2761

Distribution of Compile Times ($n_{modules} = 1025$)



- IR files were optimized and timed via `clang -O3`.

Scatter Plot of Compile Times ($n_{modules} = 1025$)



- End-2-end compile times vs. text segment sizes.
- Growth trend appears to be polynomial as a function of text segment size.

†Number of LLVM IR modules.

Preliminary Outlier Analysis ($n_{modules} = 1025$)

The largest *transformation* pass (wall) times are listed here, taken from the result of `-ftime-report`, for the longest compile times.

Table 1. Total Execution Time: 16.04 wall clock

Pass Name	Wall Time (seconds)	Percentage
InstCombine	3.38	21.1
Inliner	1.82	11.4
GVN	1.20	7.5

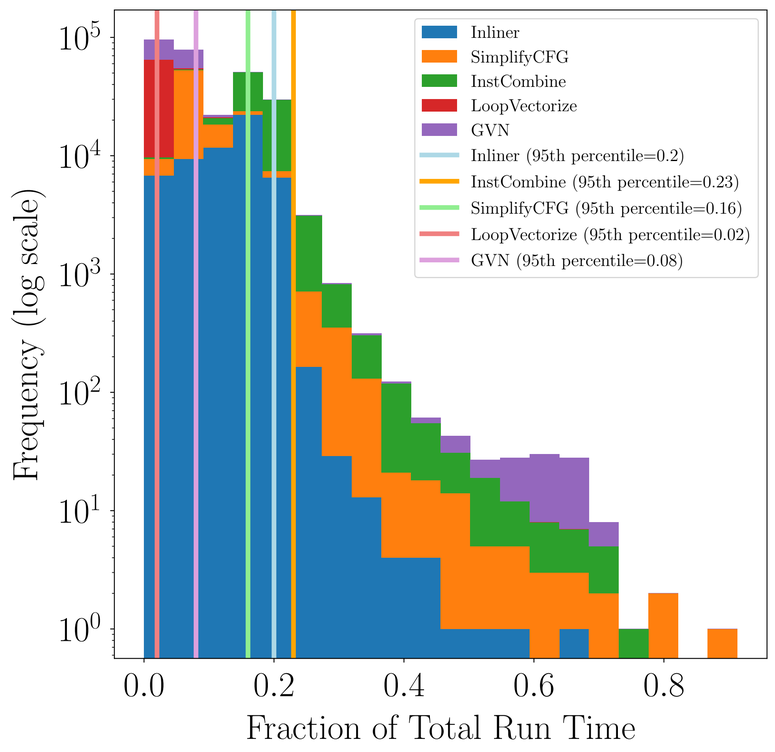
Table 2. Total Execution Time: 26.10 wall clock

Pass Name	Wall Time (seconds)	Percentage
InstCombine	4.68	17.9
Inliner	4.59	17.6
SimplifyCFG	1.49	5.7

Table 3. Total Execution Time: 45.24 wall clock

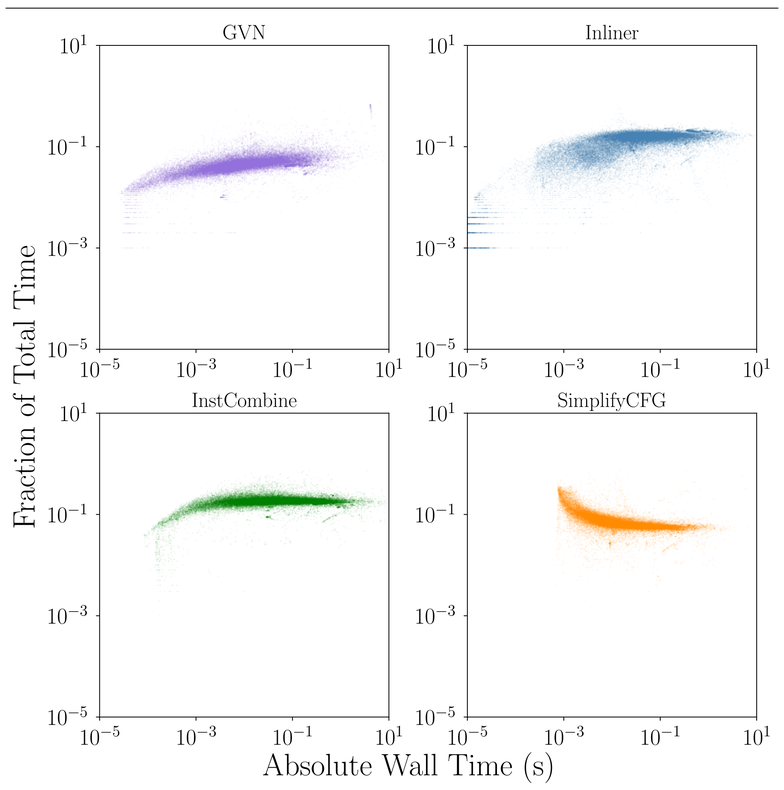
Name	Wall Time (seconds)	Percentage
Inliner	7.75	17.1
InstCombine	7.01	15.5
LoopVectorize	4.39	9.7

Relative Wall Time Distribution ($n_{modules} = 56998$)



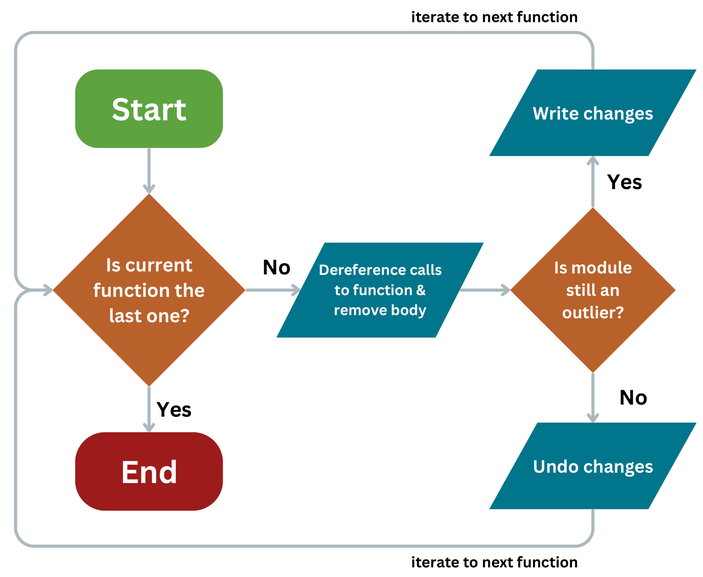
- Relative pass time in `-O3` for C++ modules.

Absolute Time vs. Relative Time ($n_{modules} = 56998$)

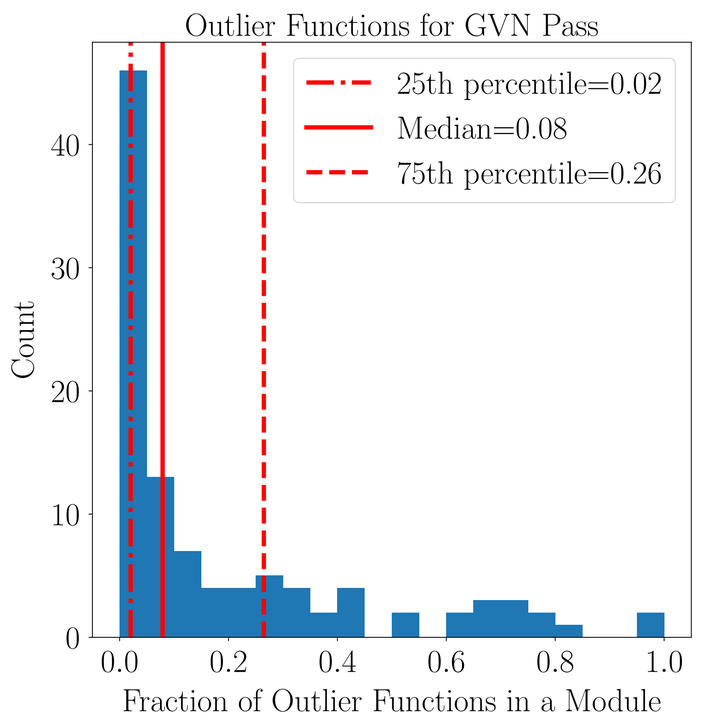


- Pass times in `-O3` for C++ modules.
- We hypothesize these trends are due to the following:
  - Instructions per BB going up with text size
  - Modules with little/no work cause horizontal banding.

Outlier Function Extraction ($n_{modules} = 1841$)



- An outlier function is defined to be a function contributing to its module being an outlier for a specific pass.



- Threshold for outlier extraction is 95th percentile for relative wall time with at least 0.005 seconds for absolute wall time to minimize noise.

Conclusion

- Compilation times appear to be non-normally distributed for all optimization levels when compiling C/C++.
- As compile time appears to grow polynomially in relation to the text segment size, outlier detection should be able to detect passes that do not conform to this trend.
- An initial outlier analysis seems to suggest specific passes encapsulate the majority of compilation time in some modules.

What do you want to see?

- Interested in specific analyses? Please contact us!

Acknowledgements

This work was in parts prepared by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-POST-862479).

This material is based upon work supported by the U.S. Department of Energy under Contract DE-FOA-0003177, S4PST: Next Generation Science Software Technologies Project.