GSoC 2024 Project Report | GCC

Improve nothrow detection for exception handling in GCC

Contributor Name: Pranil Dey
Mentors: Jan Hubicka, Martin Jambor

Project Size: Medium (~175 hours)
Project repo: patch-submission
Project proposal: Proposal

Project Abstract:

Exception handling in GCC follows the Itanium ABI, particularly the C++ exception handling ABI. This process involves creating, throwing, and finally destroying the exception object. The strategy employed is known as the Zero Cost strategy, which aims to minimize the impact on the main program execution path. It achieves this by pushing all exception handling operations into separate side tables, thus avoiding any potential negative effects on the instruction cache.

This project aims to improve the nothrow detection in GCC which will allow to eliminate the dead exception handling regions and thus optimize the intermediate code generated.

Project Goals:

The goal of this project is to make the middle end of GCC (GIMPLE) aware of the nothrow functions so as to optimize out the redundancies in the program code. This is done by extracting information out of __cxa_throw calls for GIMPLE_CALL and walking up the Control flow graph through all the basic blocks for extracting types for GIMPLE_RESX statements.

Project Results:

The information that was needed is extracted with the functions -

1. extract_types_for_call - This is a pretty simple function that detects the __cxa_throw calls made and gets the necessary type information from the typeinfo parameter of the call

2. extract_fun_resx_types - This walks through all basic blocks of a function and calls the extract_types_for_resx function to get the types in a vector and then puts them into a hash set to avoid duplicates
3. extract_types_for_resx - This is a recursive function that goes through the edges of the basic blocks and goes up the nodes pointed to by the bb->preds list. It gets the types and accumulates them into a return vector for passing onto the extract_fun_resx_types

The information gained from these is only useful if passed on further for processing which is done by the functions -

1. stmt_throw_types - This switched between the statements of GIMPLE_CALL and GIMPLE_RESX and passes it onto the function update_stmt_eh_region
2. update_stmt_eh_region - We basically update the EH regions and landing pads for a statement by walking up the EH region tree. The landing pads for the types are matched and the throw_stmt_table is updated for the function. This function is then passed on to make_edges_bb to use the now updated tree to make proper edges

Some helper functions were created along the journey as well -

1. match_lp - Used to check if a landing pad can handle any of the given exception types
2. unlink_eh_region - Used in the update_stmt_eh_region function for detaching a region from the EH tree to get reinserted later into its proper position
3. reinsert_eh_region - Used to reinsert a region after unlinking it from the EH tree
4. same_or_derived_type - Fixes the C++ type hierarchy for object oriented languages and fuses it into our exception handling processing

Contributions:

The list of all the significant contributions made -

Commit/PR	Description
Commit: 9a70651	Added extracting and processing of throw stmt exception types
Commit: c16d4a0	Added C++ inheritance for the types
Commit: d1a84d3	Added extracting types from resx stmt and integrated it to update the EH regions for a stmt
Commit: 1a736e2	Unlinking and reinserting regions functions added for removing a region from the EH tree and reinserting them in a specific place
Commit: c4ab1c5	Fixed all functions to be safe and not cause any problems with the testsuite

Future Plans:

This project is still in the testing phase and can be found in the "devel/nothrow-detection" branch of the gcc repository. We need to test, polish, and refactor the code and make it good enough to be included in the upstream.
Further implementations can be optimized for faster information gathering (fix and use something other than recursion for extracting resx types) and matching (matching the regions with the exception types needs to include ERT_ALLOWED_EXCEPTIONS)

Concluding Remarks:

I have completed almost all the goals we decided on during the community bonding period. Though it was a difficult project, I thoroughly enjoyed the work and would be thrilled to continue to contribute in the future as well. I had tried getting into open source a few times before and hit a dead end. Participating in GSoC gave me the right confidence and skill to make meaningful contributions. Now, instead of feeling overwhelmed by a big codebase, I have the skills to handle it.

Acknowledgements:

Thanks to Google, GCC, and my mentors for this opportunity. I will be forever indebted to my mentors and GCC community for their constant support and guidance.