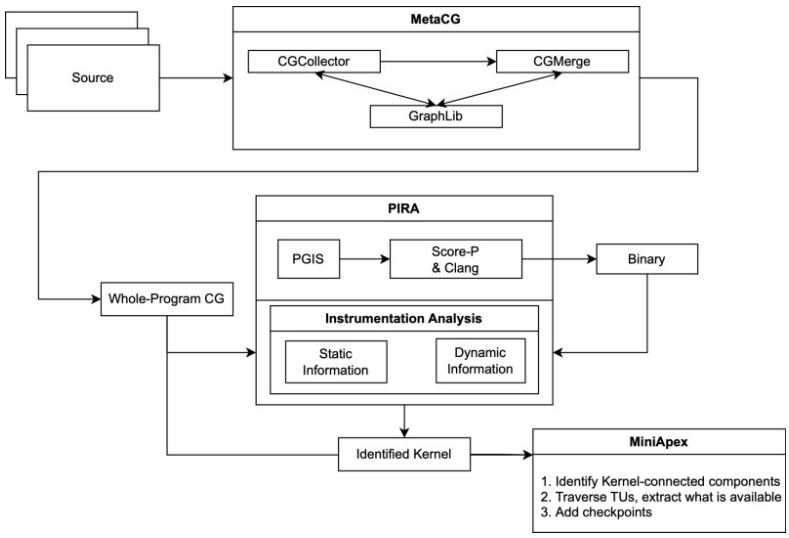
Clang Code Refactor



Project Manager
Tim Heldmann

Principal Investigator
Prof. Dr. Christian Bischof

Project Term
2023 - 2024

Clusters
Lichtenberg Cluster Darmstadt

Additional Software
Clang/LLVM

Institute
Institute of Scientific Computing

University
Technische Universität Darmstadt

Introduction

When procuring an HPC system, the system need to fulfill certain criterias, regarding performance but also energy efficiency. These are usually meassured by running industry standard benchmarks, like LINPACK or STREAM. These benchmarks however do not necesarrily reflect the way the system will be used later. Using the users codes as the benchmark would lead to a system reflects the users needs better, with the drawback that these large scientific applications take to long and are to complex to be a benchmark.

Methods

For this we designed a tool called MiniApex, that is intended to extract the small part of a user's large scientific code, that is representative for the system load generated by running the whole application. For this MiniApex makes use of the Clang compiler, to parse and analyse the sourcecode, to find the routines of the application that make up the kernel. After this identification has taken place, the code is then extracted into a new mini-app, that can then be equipped with a custom entry point to function as a benchmark. Given the entry function into the computational kernel, we recursively descent through all callees, marking them as necessary to extract. To do this across multiple sourcecode files, also called translation units, we use a whole program callgraph for this step. After collecting all functions, we step through this subset of functions a second time, this time marking all modifications to globally visible state, aswell as uses of structs and definitions as necessary to extract. This concludes the sourcecode extraction part. As Clang however operates on the expanded sourcecode, but our mini-app should represent the sourcecode before the preprocessor is run, we also hook into the preprocessor, the extract the

unexpanded form of all makros, pragmas and definitions. These are then spliced into the extracted sourcecode parts. The last remaining step is to manually write a setup routine to call this extracted mini-app.

Results

In the first part of this project the MiniApex tool was implemented and tested on the Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH) code. This code, while being comparatively small, features a diverse set of C++ features like polymorphic function calls, templates and macros. We showed, that MiniApex can extract arbitrary functions from LULESH providing a basis from which to build up a standalone mini-app.

Discussion

While this is a promising first result, the LULESH code is still relatively small compared to codes like GROMACS or OpenFoam. The main reasons are the difficulty in generating a valid call-graph, as well has handling partially instantiated templates. Additionally, C++ is a language, that gives its user many freedoms to express their needs, making it difficult to extrapolate the robustness of a tool like MiniApex based on small codes. In the followup project the MiniApex tool's robustness will be subject to continued development and optimization, to generate source-code snippets that can act as design guides for other source-code analysis and refactoring tools.

Figures

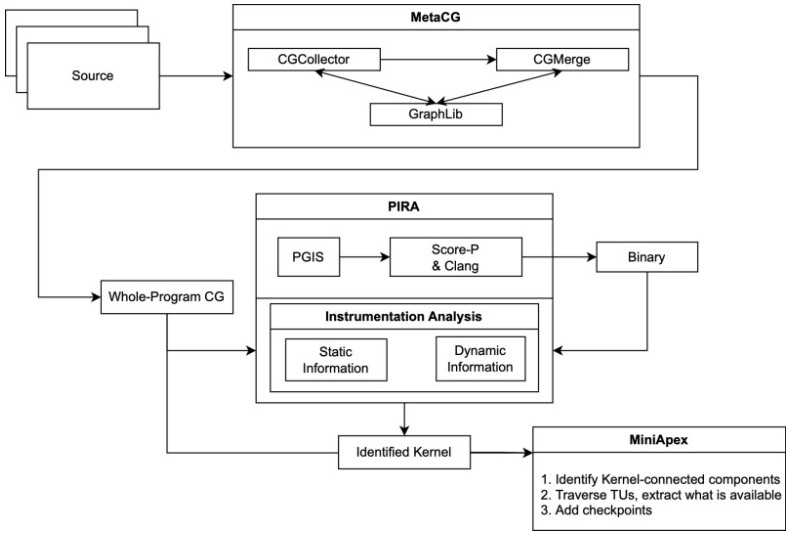


Figure 1: Example for a fully automated mini-app extraction pipeline using the MiniApex tool

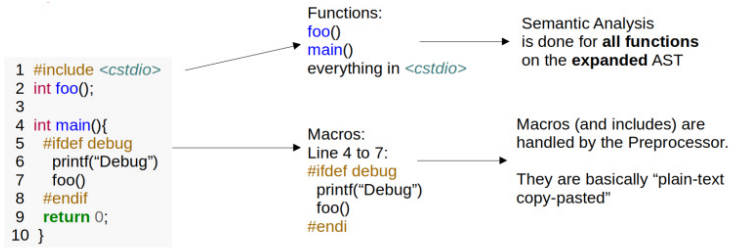


Figure 2: Separate parts of a program from the view of MiniApex

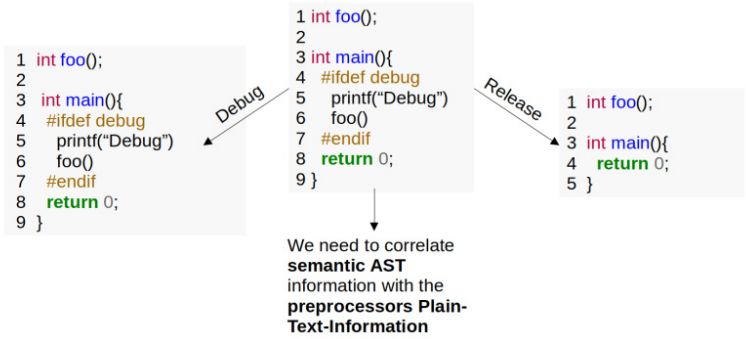


Figure 3: An example of possible influence of compile configuration on code visibility to the compiler

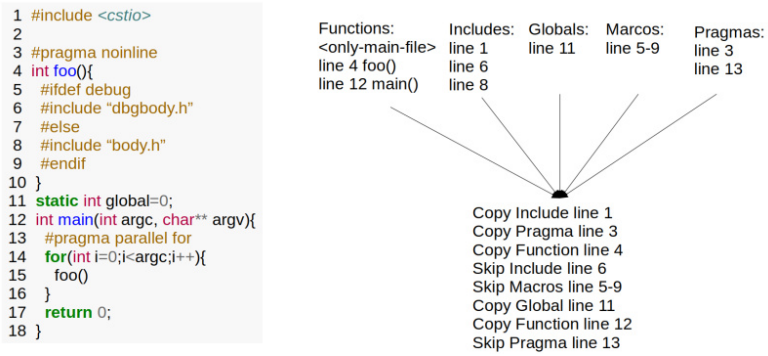


Figure 4: The five way merge based on the common source-file location information to reconstruct the program

Publications

Heldmann, T. Extracting Mini-Apps from HPC software for Total Cost of Ownership optimized systems procurement, Free and Open Source Software Developers' European Meeting / FOSDEM, Brussels, Belgium, Feb 2-3, 2024

Last Update: 2024-07-05 17:48