LLVM GSoC 2018 Proposal

PERSONAL INFORMATION

Name	Sagar Thakur
Country	India (GMT + 5:30)
School	International Information of Information Technology, Hyderabad https://www.iiit.ac.in/about/quick-facts/
Degree	Master of Technology in Computer Science and Engineering
Email	cs.sagarthakur@gmail.com sagar.thakur@students.iiit.ac.in
Contact	(+91) 8446605703

A single updater class for Dominators in LLVM

RELEVANT EXPERIENCE

Before joining IIIT hyderabad I was working at Imagination technologies where I gained 3 years of experience of working in the LLVM community. I am well aware of the source code structure and the building process. I have worked on the LLVM, LLDB and the compiler-rt projects of LLVM. My work was mainly focused on porting LLVM, LLDB and compiler-rt to support the MIPS backend. My patches and commits can be found at https://reviews.llvm.org/p/slthakur/. I also have experience working with the LLVM IR. My work on LLVM IR was mainly focused on lowering the IR to mips target assembly in the chromium subzero project. I have also worked with selection DAGs in LLVM. My patches in the subzero project can be found here.

I have gone through the relevant research papers mentioned in the comments in the source code and the dominator tree class reference diagrams. I have run the dominator tree IR tests using debug flags for the dominator tree to see how it is built.

I have studied graduate level compiler construction and optimization courses. I had also taken up a course on Advanced Problem Solving which teaches how to model data structures to solve complex computer science problems. In the Advanced problem solving course I have acquired knowledge of basic and advanced graph and tree algorithms. I have also worked on implementation of incremental dynamic depth first search algorithm [1] given in the paper by Surender Baswana and Shahbaz Khan [2] as a project in the Advanced Problem Solving course. Apart from this I have worked on various Operating System/Database System/Scripting mini projects.

[1] https://github.com/sagar-thakur/DynamicDFS
[2] https://www.cse.iitk.ac.in/users/sbaswana/Papers-published/Incr-DFS.pdf

MOTIVATION

Although I have worked previously in llvm, my work was mainly focused on the porting of MIPS processor for various LLVM projects. This is the first time I will be working on the generic part of LLVM. The area of dominator trees in LLVM is new to me and I always wanted to work on the internal data structures used in LLVM. I believe this project will be a good start for me to get into the field of compiler optimization.

OBJECTIVE OF THE PROJECT

Design and implement a new class for abstracting away how the dominator tree updates are performed when the CFG changes.

CRITERIA OF SUCCESS

1. A clean abstract updater interface for the dominator tree and the post dominator tree for the users of the dominator tree.
2. Faster incremental updates to the dominator tree and the post dominator tree. Using the new class we would be able to avoid the unnecessary changes to the Post dominator tree.

PROPOSAL

Going through the source code I have tried to understand where the changes need to be made. Here is the plan I propose to follow based on my understanding of the source code:

- Wrap all the dominator tree related classes such as the DominatorTree class, DominatorTreeAnalysis class, DefferedDominance class defined in Dominators.h and the PostDominatorTree class defined in PostDominators.h with a single updater class.
- Make code changes in the places that use the dominator tree such as the LoopSimplify, BreakCriticalEdges, LoopUnroll etc. Now we will use the single updater class.
- To make the updation faster I will design an algorithm to prune unnecessary changes to the Post Domination Tree.
- Design new strategies to perform lazy updates.

TIMELINE

I have tried to come up with a preliminary timeline which could be followed to complete the project. I am open to any change in the timeline according to the mentor.

Duration	Description
April 23	Acceptance of students proposals

April 23 to April 29 (Community Bonding Period)	● Initial discussion about requirements for various modules with mentor. ● Set up of weekly schedule for updates via skype/email/chat. ● Explore source code with current state of repository.
April 30 to May 6 (Community Bonding Period)	● Discuss requirements with mentor. ● Continue understanding current codebase. ● Start designing a simple sketch for the updater class.
May 7 –to May 14 (Community Bonding Period)	● Finalize implementation details. ● Start implementation of the updater class.
May 15 to May 21	● Finish implementation of the updater class and submit patch for it. ● Address the review comments on the patch and commit the changes.
May 22 to May 28	● Use the new updater object to the existing code which was working directly with the dominator / deferred dominator tree. ● Write and evaluate test cases. ● Submit patch the changes and the test cases.
May 29 to June 10	● Address the review comments on the patch. ● Add more tests if needed. ● Commit the changes after acceptance of the patch.
June 11 to June 15 (Phase 1 Evaluation)	● Add documentation if needed. ● Discuss progress with mentor.
June 16 to June 24	● Start design and visualize the algorithm to prune unnecessary PostDomTree updates based on updates to the DomTree. ● Finalize design of algorithm with mentor.

June 25 to July 1	● Implement the algorithm. ● Write and evaluate test cases. ● Submit patch for it. ● Address review comments and commit patch after acceptance.
July 2 to July 8	● Compare performance metrics for previous implementation and current implementation.
July 9 to July 13 (Phase 2 Evaluation)	● Add documentation if needed. ● Discuss progress with mentor.
July 14 to July 20	● Continue to compare performance metrics for previous implementation and current implementation. ● Start exploring more strategies for performing lazy updates.
July 21 to July 27	● Discuss with mentor about various strategies for lazy updates. ● Implement the best strategy discussed with mentor. ● Write and evaluate test cases.
July 28 to Aug 5	● Write and evaluate more test cases if needed. ● Submit patch and commit on acceptance. ● Complete any unfinished task, take up some new tasks that require to be implemented as per the mentor's directive.
Aug 6 to Aug 14	● Final week: Submit final product and wait for mentor's evaluation.

Proposal for improvisation of debugging in optimized code

RELEVANT EXPERIENCE

Before joining IIIT hyderabad I was working at Imagination technologies where I gained 3 years of experience of working in the LLVM community. I am well aware of the source code structure and the building process. I have worked on the LLVM, LLDB and the compiler-rt projects of LLVM. My work was mainly focused on porting LLVM, LLDB and compiler-rt to support the MIPS backend. My patches and commits can be found at https://reviews.llvm.org/p/slthakur/. I also have experience working with the LLVM IR. My work on LLVM IR was mainly focused on lowering the IR to mips target assembly in the chromium subzero project. I have also worked with selection DAGs in LLVM. My patches in the subzero project can be found at here.

To get a feel of the debugify utility I applied the debugify utility on the test case 2012-07-18-LimitReassociate.ll. I found that there was a debug info loss error reported by the check-debugify utility. By using the print-after-all option of opt I found that there was a bitcast instruction inserted to the code without a debug loc. To drill down the bug I debugged the loop strength reduce optimization pass to find that the bitcast code was being inserted by the InsertNoopCastTo function of the ScalarEvolutionExpander. I am not sure how to teach the function to add the debug loc, but I hope to discuss this with the mentor and solve this issue.

MOTIVATION

Although I have worked previously in llvm, my work was mainly focused on the porting of MIPS processor for various LLVM projects. This is the first time I will be working on the generic part of LLVM. I always wondered how debug information was propagated during compilation and I always wanted to work on the internals in LLVM. This project will also allow me to tweak the optimization passes which I believe will be a good start for me to get into the world of LLVM .

OBJECTIVE OF THE PROJECT

To find out as many debug information bugs as possible in LLVM's hotspot optimization passes and teach these passes to preserve the debug information.

CRITERIA OF SUCCESS

- Develop an automated way to run the debugify and check-debugify utility on each optimization pass given a test and calling this mode as debugify-each.
- Run this utility on each of the optimization tests in the current llvm test suite and prepare an excel sheet of debug info loss bugs.
- Generate more tests based on the information we found in the excel sheet above targeting specific passes.
- Fix the debug info loss bugs uncovered in the generated tests.

PROPOSAL

I propose the following plan of action for the completion of the project:

- First I would work on the debugify-each mode for the opt tool. I read on the mailing list that such a utility is required and indeed this utility will simplify out task of tracing out debug info loss bugs.
- Then I would use debugify-each mode on all the optimization test cases currently in the llvm test suite to identify hotspot areas of optimization where improvements need to be made.

- Then I would generate new test cases targeting these hotspot areas.
- Teach the optimization passes to preserve the debug info for each test case at a time.

TIMELINE

I have tried to come up with a preliminary timeline which could be followed to complete the project. I am open to any change in the timeline according to the mentor.

Duration	Description
April 23	Acceptance of students proposals
April 23 to April 29 (Community Bonding Period)	● Initial discussion about requirements for various modules with mentor. ● Set up of weekly schedule for updates via skype/email/chat. ● Explore source code with current state of repository.
April 30 to May 14 (Community Bonding Period)	● Discuss requirements with mentor. ● Read LLVM documents for implementing new passes, opt tool etc.
May 15 to May 28	● Implementation of the debugify each option for the opt tool. ● Implement a way for the debugify each tool to generate the statistics on debug info loss. ● Submit patch and address review comments.
May 29 to June 10	● Apply the debugify each utility in each of the llvm test suite optimization passes. ● Prepare an excel sheet on the data generated by the debugify each utility.
June 11 to June 15 (Phase 1 Evaluation)	● Add documentation if needed. ● Discuss progress with mentor.
June 16 to June 24	● Discuss with mentor about various hotspot areas which need to be covered. ● Come up with test cases for the hotspot areas uncovered.
June 25 to July 8	● Start teaching the optimization passes how to preserve debug information. ● Submit patches and address review comments.

July 9 to July 13 (Phase 2 Evaluation)	● Add documentation if needed. ● Discuss progress with mentor.
July 14 to July 27	● Write and evaluate more test cases. ● Incrementally keep adding new test cases and fix for their debug information loss.
July 28 to Aug 5	● Complete any unfinished task, take up some new tasks that require to be implemented as per the mentor's directive.
Aug 6 to Aug 14	● Final week: Submit final product and wait for mentor's evaluation.

AVAILABILITY

My semester exams finish by the 26th of April allowing me to get started right away after that. I have no prior commitments during the months of May, June and July. My college reopens in the 1st week of August, but initially the load would be quite less and by that time I would be able to complete most of the work. I would be able to work 6+ hours on a daily basis. Thus I will be able to devote a minimum of 42 hours per week for the project.

A NOTE OF THANKS

I would like to thank the llvm community for giving me this opportunity to write a proposal. I look forward to receiving feedback from those reviewing this document, and would be glad to discuss/change accordingly.