Homework 4 - Writing LLVM Passes

CSIE 5054 - Advanced Compiler Design
National Taiwan University

Total Points:	$8 + 4$
Release Date:	2024/11/29 08:00
Due Date:	2024/12/20 23:59
TA Email:	llvm@csie.ntu.edu.tw

Contents

1 Announcement

In case you encounter technical issues with Homework 4, kindly consult online resources initially, as most problems are likely related to your local environment. If challenges persist, reach out to our TAs following these guidelines:

- Title your email [AC-HW4][Summary-Of-Your-Issue]. Please note that we will **NOT** receive emails in other formats as we have applied filters to our email system.
- Provide detailed information about your computer, including the operating system.
- Outline the methods you attempted previously, the resources you consulted, the steps you followed, and the results of your efforts.
- Note that ambiguous requests, such as attaching screenshots without proper descriptions, will not be answered. **Such emailing will lower your priority.**
- For guidance on how to formulate effective technical inquiries, please refer to How To Ask Questions The Smart Way. This resource can help you structure your questions to get quicker, more precise responses from the TAs.

2 Problem 1

In this homework, you will implement an optimization pass in LLVM to improve program performance while maintaining correctness. You will need to carefully design, implement, test, and validate your optimization.

2.1 Requirements

2.1.1 Core Implementation (8 points)

You must implement one optimization pass thoroughly and effectively. Requirements for the pass:

- Design and document your optimization algorithm in detail

- Implement the pass using the LLVM framework
- Create comprehensive test cases demonstrating performance improvements
- Prove the correctness of your optimization rigorously

2.1.2 Optional Bonus Implementation (up to 4 points)

You may implement one additional optimization pass for bonus points.

2.1.3 Available Optimization Passes

The following topics represent optimization opportunities in LLVM. These are guidelines for exploration - you are expected to research, design, and implement your own specific optimization strategies within these topics or propose new ones.

1. **Global Code Placement**
    - Implement profile-guided basic block ordering
    - Design branch prediction heuristics
    - Optimize instruction cache behavior
    - Handle branch alignment and fall-through optimization
    - Consider hot/cold path separation
2. **Loop-based Optimizations**
    - Implement loop unrolling with cost-based decisions
    - Design software pipelining for instruction-level parallelism
    - Support loop unswitching for conditional optimization
    - Handle loop invariant code motion
    - Handle lazy code motion
    - Consider other loop-based optimization opportunities
3. **Function-based Optimizations**
    - Implement context-sensitive function inlining
    - Support function outlining for code size reduction
    - Implement devirtualization for indirect calls
    - Consider other inter-procedural optimization opportunities
4. **Peephole Optimizations**
    - Design pattern-matching framework
    - Implement automatic pattern generation
    - Support verification of transformations
    - Consider cost-based pattern selection
5. **Custom Optimization**
    - Other optimization techniques
    - Theoretical foundations
    - Performance improvement focus

Before selecting a topic, thoroughly understand:

- The theoretical foundations and algorithms
- Implementation complexity and resource requirements
- Potential performance impact and trade-offs
- Verification and correctness requirements

2.2 Deliverables

2.2.1 Technical Report

Submit a technical report in **IEEE two-column** format containing:

1. **Algorithm Design:**
   - Detailed explanation of the optimization strategy
   - Theoretical analysis of potential improvements
   - Implementation considerations and challenges
   - Code examples illustrating the transformation
2. **Implementation Details:**
   - LLVM pass implementation methodology
   - Key data structures and algorithms used
   - Integration with the LLVM optimization pipeline
   - Handling of edge cases and special conditions
3. **Experimental Evaluation:**
   - Description of test cases and benchmarks
   - Performance measurements and analysis
   - Comparison with unoptimized code
   - Statistical significance of improvements
4. **Correctness Proof:**
   - Formal or informal proof of correctness
   - Invariant preservation
   - Edge case analysis
   - Testing methodology and results

2.3 Testing Requirements

2.3.1 Test Cases and Benchmarks

For each optimization pass:

- Provide a representative set of test cases that demonstrate your optimization's effectiveness
- Include test cases that cover:
  - Simple cases to verify basic functionality
  - Complex scenarios that demonstrate real-world applicability
  - Edge cases that verify robustness
  - Cases where the optimization should and should not be applied

- Select benchmarks that are:
  - Representative of typical use cases
  - Meaningful for your chosen optimization
  - Able to demonstrate significant performance impact

2.3.2 Performance Evaluation

- Focus on relevant metrics for your optimization:
  - Execution time for runtime optimizations
  - Memory usage for space optimizations
  - Code size for size optimizations
  - Compilation time
- Provide clear before/after comparisons
- Include analysis showing the significance of improvements

2.4 Reproducibility Requirements

2.4.1 Experimental Environment

All implementations and experiments must be reproducible on the **ws1**:

- Document any specific environment variables or configurations needed
- Ensure all dependencies are available on **ws1** or properly documented for installation
- Test your complete submission on **ws1** before final submission

2.4.2 Important Notes

- Your grade will partially depend on our ability to reproduce your results on **ws1**
- All performance measurements must be conducted on **ws1** for fair comparison
- If your implementation requires specific hardware features, verify their availability on **ws1**
- If TAs encounter any reproducibility issues with your submission, you may be required to demonstrate your results on-site using your own computer

2.5 Submission Guidelines

2.5.1 Code Submission

- Submit source code for all optimization passes
- Include build instructions and dependencies
- Provide scripts for running tests
- Document any external libraries or tools used

2.5.2 Report Format

- Use **IEEE two-column** format in LaTeX
- If implementing multiple passes, clearly indicate:
  - Which pass is for core implementation (8 points)
  - Which pass is for bonus points (4 points)
- Include clear sections for each optimization pass
- Provide citations for referenced work

2.6 Submission Instructions

1. Prepare your submission files:
   - Technical report in PDF format named `report.pdf`
   - Source code directory containing all implementation files
   - Test cases and evaluation scripts
   - `README.md` file with detailed setup and reproduction instructions
2. Create a zip file named `[your_student_id].zip` (e.g., `r12345678.zip`)
3. Submit the zip file to NTU COOL before the deadline
4. The zip file structure should be like:

```
r12345678.zip
├── report.pdf
├── README.md
├── src/
│   ├── [your optimization pass files]
│   └── [other source files]
└── tests/
    ├── [test cases]
    └── [evaluation scripts]
```

2.6.1 README.md Requirements

Your `README.md` must include:

- **Environment Setup**
  - Required LLVM version and installation instructions
  - All dependencies and their versions
  - Required environment variables or system configurations
- **Build Instructions**
  - Step-by-step guide to build your optimization pass
  - Commands to compile and link with LLVM
  - Any special build flags or configurations
- **Running the Pass**
  - Commands to run your optimization pass

  - Description of any command-line options

- **Testing Instructions**
  - How to run the test suite
  - Commands to reproduce performance measurements

Note: Submissions with missing components will **NOT** be graded.

2.7 Grading Criteria

2.7.1 Core Implementation (8 points)

- Algorithm Design and Documentation (3 points)
  - Theoretical foundation and motivation (1.5 points)
  - Optimization strategy and analysis (1.5 points)
- Testing and Correctness Analysis (3 points)
  - Test cases design and benchmarks selection (1 point)
  - Invariant analysis and preservation (1 point)
  - Proof strategy and correctness discussion (1 point)
- Performance Enhancement (2 points)
  - Performance enhancement demonstration (1 point)
  - Analysis of improvement significance (1 point)

  Note: Performance Enhancement section will only be evaluated if the optimized program maintains semantic correctness

2.7.2 Bonus Implementation (up to 4 points)

Following the same criteria distribution as above, scaled to 4 points total.

2.7.3 DO

- Implement each optimization pass from scratch
- Write your own analysis and transformation logic
- Use LLVM's APIs and data structures
- Write your implementation in **C++** to leverage the full power of LLVM's APIs

2.7.4 DON'T

- Copy existing LLVM pass implementations
- Submit code from other sources without proper attribution
- Submit the whole llvm-project.

2.8 Additional Notes

- **Start early** to allow time for testing and refinement
- Consult LLVM documentation for implementation details
- Consider both compile-time and run-time performance
- Document any assumptions or limitations
- If you encounter implementation difficulties, you can still submit a detailed write-up to demonstrate your understanding and effort
- Good luck with your homework and final exam :)

3 Plagiarism and Academic Integrity

It is important to adhere to the university's academic integrity policy while completing this assignment. Please keep the following in mind:

1. **Do Not Share Your Code**: All submissions must be your own work. Sharing your code with others or obtaining code from others, including online resources, is considered plagiarism and will be treated as academic misconduct.
2. **Use of External Resources**: You are encouraged to textbooks, lecture notes, and official documentation to assist your understanding. However, directly copying code or solutions from external sources (e.g., GitHub, StackOverflow, or similar) without attribution is not allowed.
3. **Collaboration Guidelines**: You may discuss general concepts and strategies with classmates, but all coding and detailed design decisions must be completed **independently**.
4. **Consequences of Plagiarism**: Plagiarism, cheating, or any form of academic dishonesty will result in a zero on the assignment, and may lead to further disciplinary action as per university regulations.

4 Additional Resources

- LLVM User Guide: `https://llvm.org/docs/UserGuides.html`
- LLVM Legacy Pass Manager Manual: `https://llvm.org/docs/WritingAnLLVMPass.html`
- LLVM New Pass Manager Manual: `https://llvm.org/docs/NewPassManager.html` and `https://llvm.org/docs/WritingAnLLVMNewPMPass.html`
- Writing an LLVM Pass: 101: `https://llvm.org/devmtg/2019-10/slides/Warzynski-WritingAnLLVMPass.pdf`
- Extending LLVM: Custom Passes and Backend Development: `https://slaptijack.com/programming/extending-llvm-custom-passes-and-backend-development.html`
- LLVM for Grad Students: `https://www.cs.cornell.edu/~asampson/blog/llvm.html`
- IEEE Two-column Format: `https://www.date-conference.com/format.pdf`