Code Optimization For Compiler Design

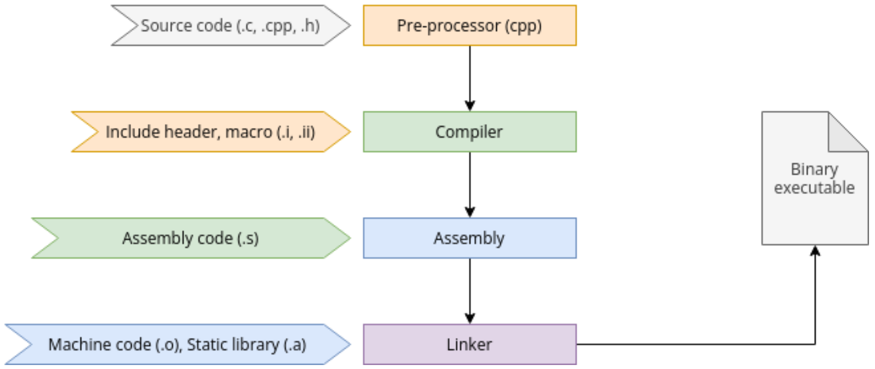
k.Vishnu vardhan(192224265)
D.Deepak(192210646)
M.Jyothieswar(192211468)

Introduction to Code Optimization for Compiler Design

Code optimization is the process of modifying code to improve its performance or reduce its resource usage.

The main goals of code optimization are to make the code run faster, use less memory, and produce more efficient machine code.

Compiler design plays a crucial role in implementing code optimization techniques to enhance program efficiency.



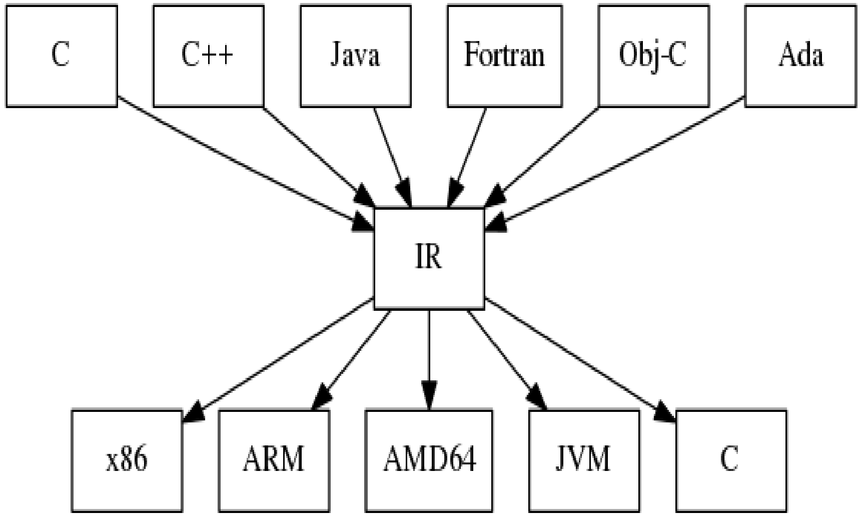
Source code (.c, .cpp, .h)
Pre-processor (cpp)
Include header, macro (.i, .ii)
Compiler
Assembly code (.s)
Assembly
Machine code (.o), Static library (.a)
Linker
Binary executable

Importance of Code Optimization in Compiler Design

Code optimization helps in reducing the execution time of programs and improving overall system performance.

It allows compilers to generate more efficient code that consumes fewer system resources.

Effective code optimization can lead to significant improvements in program speed and responsiveness.

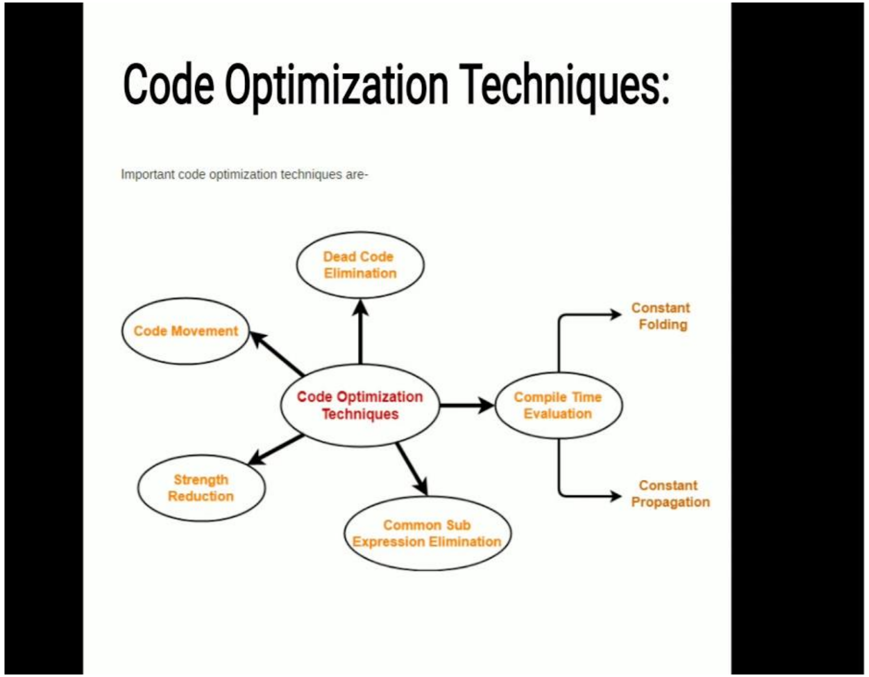


Common Code Optimization Techniques

Loop unrolling: Replicating loop bodies to reduce loop overhead and improve instruction-level parallelism.

Constant folding: Evaluating constant expressions at compile time to eliminate unnecessary computations.

Dead code elimination: Removing unreachable or redundant code to reduce program size and improve execution speed.



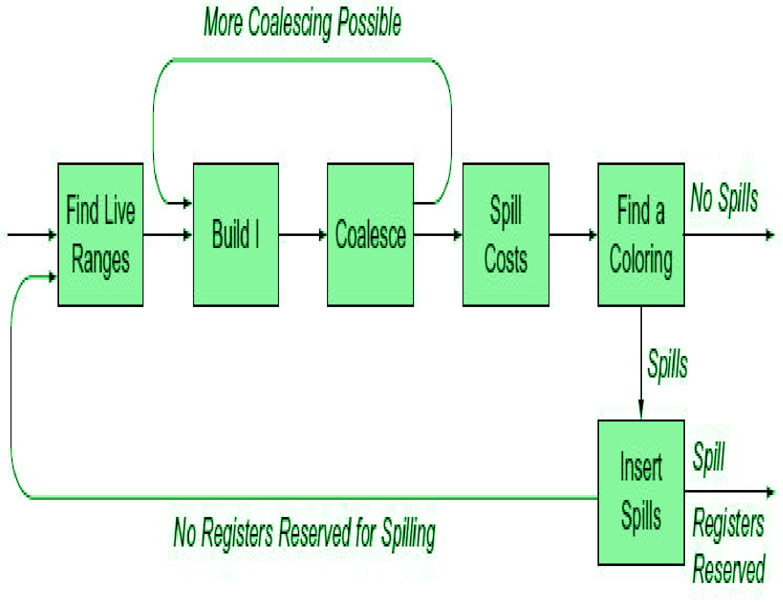
Code Optimization Techniques:
Important code optimization techniques are-
Dead Code Elimination
Code Movement
Constant Folding
Code Optimization Techniques
Compile Time Evaluation
Strength Reduction
Constant Propagation
Common Sub Expression Elimination

Advanced Code Optimization Strategies

Register allocation: Assigning variables to processor registers to minimize memory accesses and improve performance.

Inline expansion: Replacing function calls with the actual function body to reduce overhead and improve code locality.

Data flow analysis: Analyzing the flow of data through a program to optimize memory access patterns and reduce dependencies.

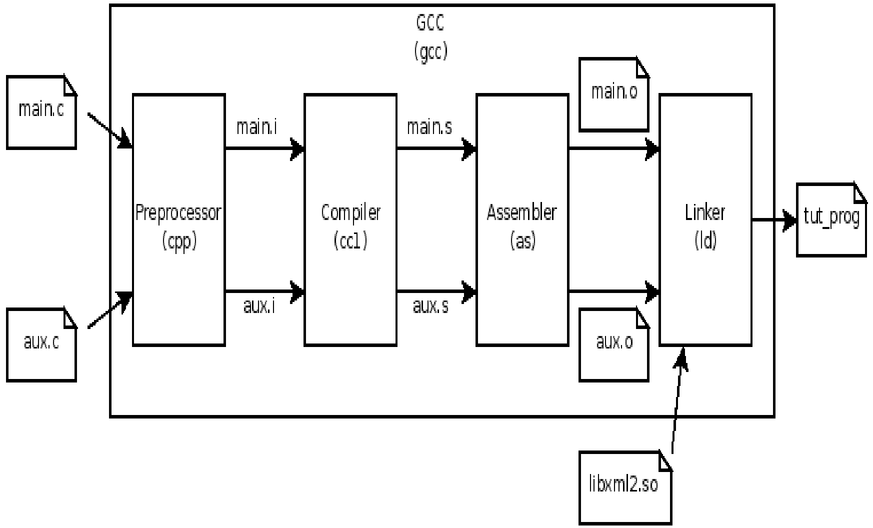


Challenges in Code Optimization for Compiler Design

Balancing between code size and execution speed: Some optimization techniques may increase code size, leading to potential trade-offs.

Platform-specific optimizations: Tailoring code optimizations for different architectures and instruction sets can be complex.

Handling complex code patterns: Optimizing code with intricate control flow or data dependencies can be challenging for compilers.

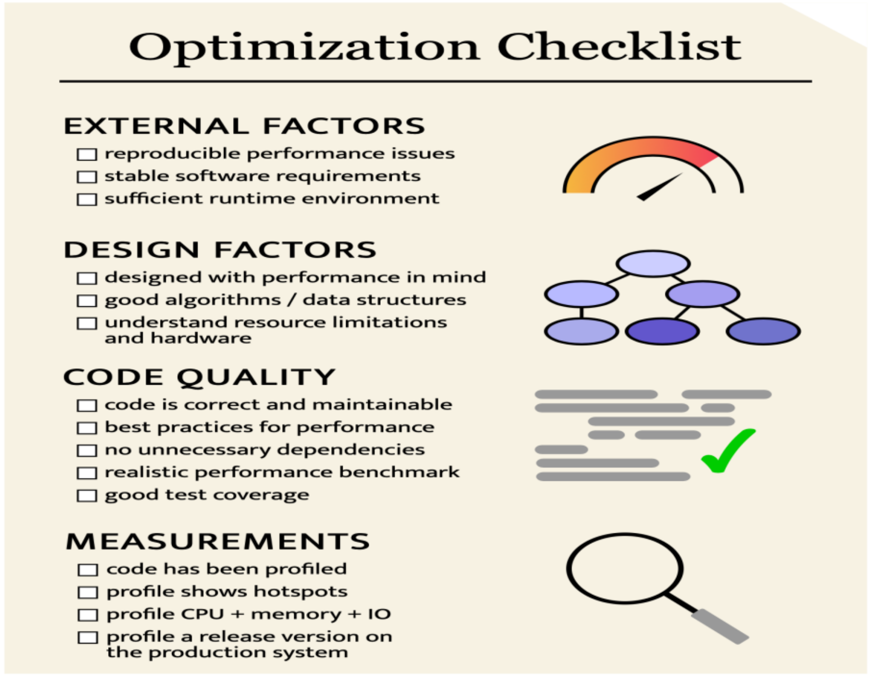


Tools and Techniques for Code Optimization

Profiling tools: Identifying performance bottlenecks in code to prioritize optimization efforts.

Compiler flags: Using compiler-specific flags to enable optimization levels and control the optimization process.

Manual optimization: Writing optimized code by hand to achieve specific performance goals that automated tools may not address.



Optimization Checklist

EXTERNAL FACTORS
- ☐ reproducible performance issues
- ☐ stable software requirements
- ☐ sufficient runtime environment

DESIGN FACTORS
- ☐ designed with performance in mind
- ☐ good algorithms / data structures
- ☐ understand resource limitations and hardware

CODE QUALITY
- ☐ code is correct and maintainable
- ☐ best practices for performance
- ☐ no unnecessary dependencies
- ☐ realistic performance benchmark
- ☐ good test coverage

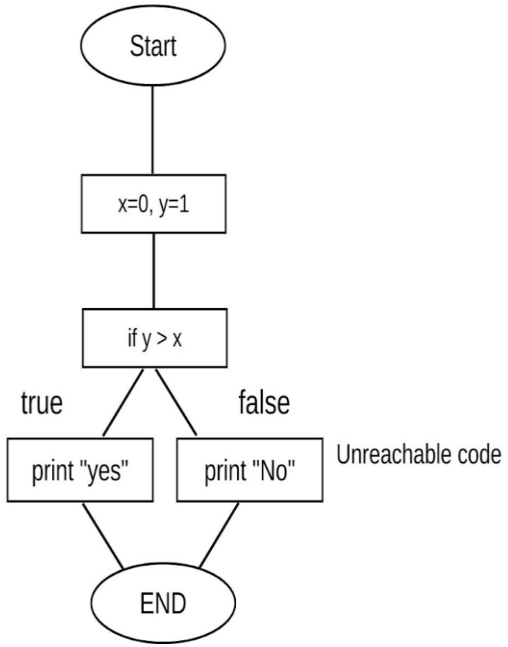
MEASUREMENTS
- ☐ code has been profiled
- ☐ profile shows hotspots
- ☐ profile CPU + memory + IO
- ☐ profile a release version on the production system

Best Practices for Code Optimization in Compiler Design

Understand the underlying hardware architecture to tailor optimizations for specific platforms.

Continuously profile and benchmark code to identify performance bottlenecks and prioritize optimization efforts.

Consider the trade-offs between code size, execution speed, and maintainability when applying optimization techniques.



thank you!