Producing Optimal Code Using the Unharnessed Power of Esoteric Language Compilers

Boróka Aradi, Levente Cantwell, András Szabó, Richárd Szigeti

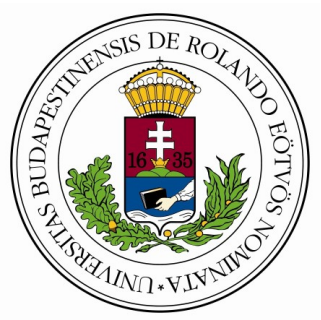

Table of Contents

Introduction

- **Problem:** Developers rely on hardware optimization too much.
- **Insight:** Languages like Brainfuck and ><> (fish) prove that even unconventional syntax can express meaningful logic.
- **Claim:** Esoteric languages offer unique optimization strategies such as branch avoidance, stack minimization, and hyper-metaprogramming.
- **Goal:** Identify and refine overlooked optimization techniques from esoteric languages to improve conventional compiler performance.

Introduction - Example


```
++++++++++[>++++++>++++++++++>
+++++++>++++>+++>+<<<<<<-]>+++
+++.>++.+++++++..+++.>>.>-.<<-
.<.+++.------.--------.>>>+.>-.
```

Listing 1: Example code of Hello World! written in Brainfuck

```
0voa                                    ~/?=0:\
 voa                oooo'Buzz'~<         /
 >1+:aa*1+=?;::5%:{3%:@*?\?/'zziF'oooo/
 ^oa                      n:~~/
```

Listing 2: Example code of FizzBuzz written in ><>

Methodology

- **Languages Studied:**
  - Icon
  - Whitespace
  - Brainfuck
  - Sparrow
- **Comparison:** Benchmarked against conventional languages (Java, C++, Python).
- **Our Technique:** Poisson-Steve.

Figure: The phases of Poisson-Steve technique

Methodology - Phase 1: Pre-processing

- **Step 1:** Minimize branching using special inbuilt values.
- **Step 2:** Minimize stack depth using various techniques.
- **Step 3:** Maximize bit-level and computing parallelism trough bit-level structures and code partitioning.

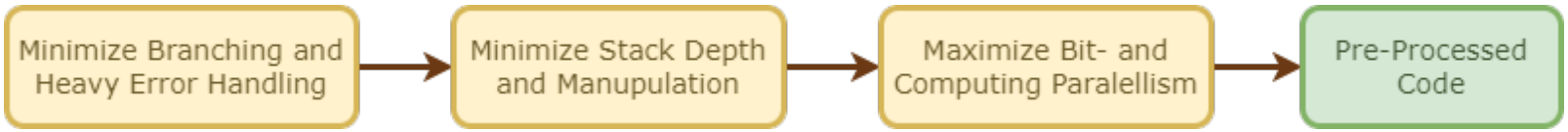

Figure: The steps of first phase

Methodology - Phase 2: Hyper-metaprogramming

- Apply hyper-metaprogramming using the pre-processed code as input.
- Move program parts from the run- into the compile-time environment.
- Produce the optimized code as output.

Figure: The steps of second phase

Results - Runtime

Program	Poisson-Steve	Optimizer 1	Optimizer 2
Hello World	20 ms	20 ms	21 ms
2000 Line Program	483 ms	546 ms	573 ms
2 Million Line Program	3246 ms	4976 ms	5647 ms

Table: Runtime comparison across compiler techniques.

Results - Memory usage

Program	Poisson-Steve	Optimizer 1	Optimizer 2
Hello World	0.6 MB	0.6 MB	0.6 MB
2000 Line Program	96.2 MB	102.4 MB	110.2 MB
2 Million Line Program	1978.2 MB	2034.8 MB	2125.4 MB

Table: Memory usage comparison across compiler techniques.

Results - Maximum stack depth

Program	Poisson-Steve	Optimizer 1	Optimizer 2
Hello World	4	4	4
2000 Line Program	96	162	174
2 Million Line Program	754	1226	1298

Table: Peak stack depth across compiler techniques.

Acknowledgment

We would like to thank the Department of Computer Science at Eötvös Loránd University and our teacher Ábel Szauter for providing resources and support during this research.

Bibliography I

 **Harry Xu, Nick Mitchell, Matthew Arnold, Atanas Rountev, and Gary Sevitsky.** "Software bloat analysis: Finding, removing, and preventing performance problems in modern large-scale object-oriented applications."

 **Daniel Temkin.** "The Less Humble Programmer."

 **Kenneth Walker and Ralph E. Griswold.** "An optimizing compiler for the Icon programming language."

 **Rutger Witmans and Vadim Zaytsev.** "Perfecting Nothingness by Refactoring Whitespace."

 **Lucian Teodorescu and Rodica Potolea.** "Compiler Design for Hyper-metaprogramming."

 **Jeremy Singer and Steve Draper.** "Let's Take Esoteric Programming Languages Seriously."

Bibliography II

 **Sang-Woo Jun.** “50,000,000,000 Instructions Per Second: Design and Implementation of a 256-Core BrainFuck Computer.”