UE18CS390A – Capstone Project Phase – 1

Project Progress Review #3

Project Title : Automated Tool for Source Code Optimization
Project ID : PW22NSK01
Project Guide : Prof. N S Kumar
Project Team : Khushei Meghana Meda, Sriram Subramanian, Shashank Vijay, Adithya Bennur

Agenda

1. Feedback from previous review
2. Project Progress
3. Source code generator
4. Demonstration of one simple example per optimization
5. Demonstration of combined optimizations
6. Demonstration of performance metrics

Suggestions from previous reviews

Suggestions-

- Specify the number of optimizations
- Create a set of test cases
- Implement a web based GUI for the tool

Feasibility and progress-

- Aim to implement at least 10 optimizations
- Tested every optimization with 20-50 unique test cases

Project Progress

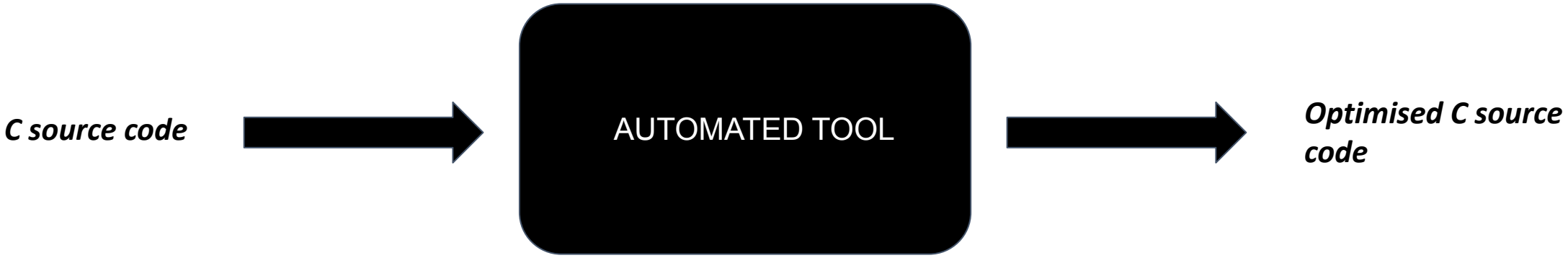
- We have been progressing at a fairly consistent pace

- We have implemented the following optimizations-
  - Loop unrolling (in progress)
  - Function inlining
  - If-else to switch
  - Compile-time initialization
  - Tail recursion elimination (in progress)
  - Loop jamming (in progress)

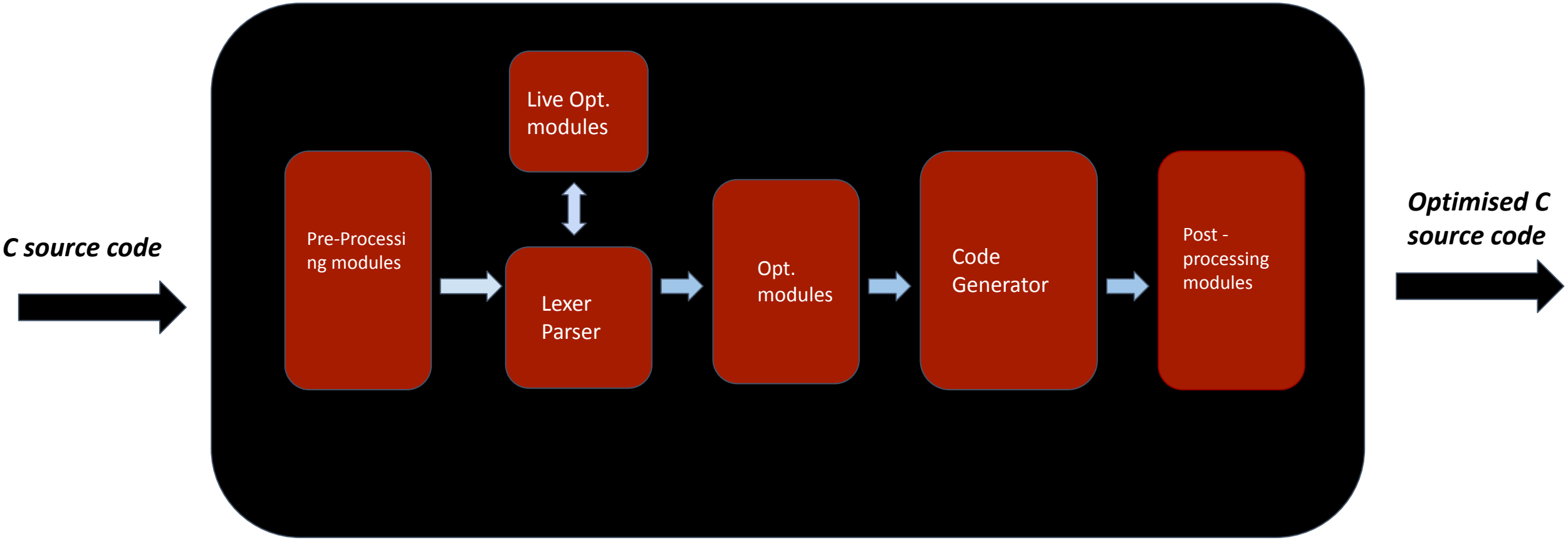
- We have developed a basic web based GUI for the user.

- We hope to refine & complete the above rules and implement 4 more rules.

Block Diagram



Block Diagram - 2



Loop Unrolling

For range that can be determined at compile time : (A complete standard for loop)

- Partial Unrolling (by a configurable factor)
- Full Unrolling

For range governed by variables whose values are determined at runtime (Not implemented by compilers).

Decreases number of comparisons done in the loop from n ----> n/2.

Rigorous testing for 40+ test cases.

Function Inlining

- Inline all non-recursive functions
- Handle function signatures of various kinds
- Inline nested functions if possible
- Add a hash value following temporary variables
- Retain the function definition

Tail Recursion Elimination

- Detects tail recursive calls.
- Sets the parameters for the next in the line function frame.
- Replaces the TR call with a ‘goto’ that directs the control to the function beginning.
- Mostly overwrites the existing function frame.
- Saves a considerable amount of stack space.

If-to-Switch

- Identifying individual chains of if-else if-else
  - Chain could end in else or else if
  - Handling nested cases

- Checking if conditions are in any of the following forms
  - var == int
  - var == char
  - lower_bound <= var <= upper_bound
  - lower_bound <= var < upper_bound
  - lower_bound < var <= upper_bound
  - lower_bound < var < upper_bound

- Semantics in conditions

- Reordering logically non-sequential range cases

Compile Time Initializations

Detects assignment of array members in a loop, and instead, generates the series and initializes values at compile time.

- Constant range based loops in which assignments occur
- Single dimensional arrays (As of now)
- Nested for loops are not considered (As of now)
- Tested for 10+ test cases.

Loop Jamming

Detects for loops with similar ranges and no data dependencies and fuses the two loops into a single one maintaining syntactic correctness.

- Reduces number of comparisons from 2n-->n.
- Currently able to fuse for loops with exactly same range assumed to be data independent.
- Data dependency checking is currently not handled.
- Nested loop jamming is currently not handled

Design Approach

Our approach is incremental and iterative.

This approach has been chosen to maintain a balance between the time spent in feasibility study and implementation.

Benefits: Parallelism in implementation of various components.

Drawbacks: Requirement of dedicated time for integration of various components.

Source Code Generation

Sample input :

['int', 'a', '=', [['b', '*', 'c'], '-', 'd'], ';']

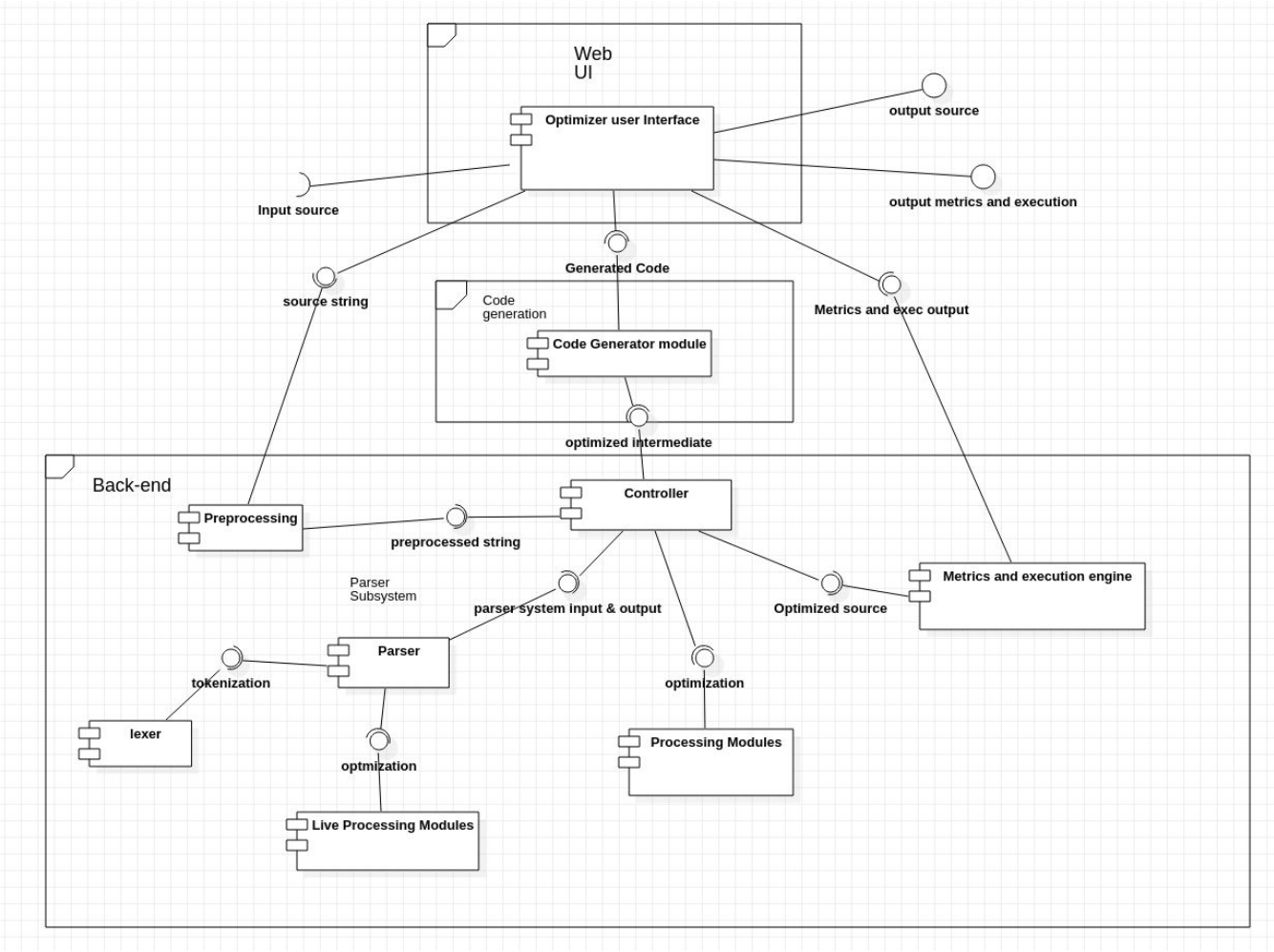
Sample output :

int a=b*c-d;

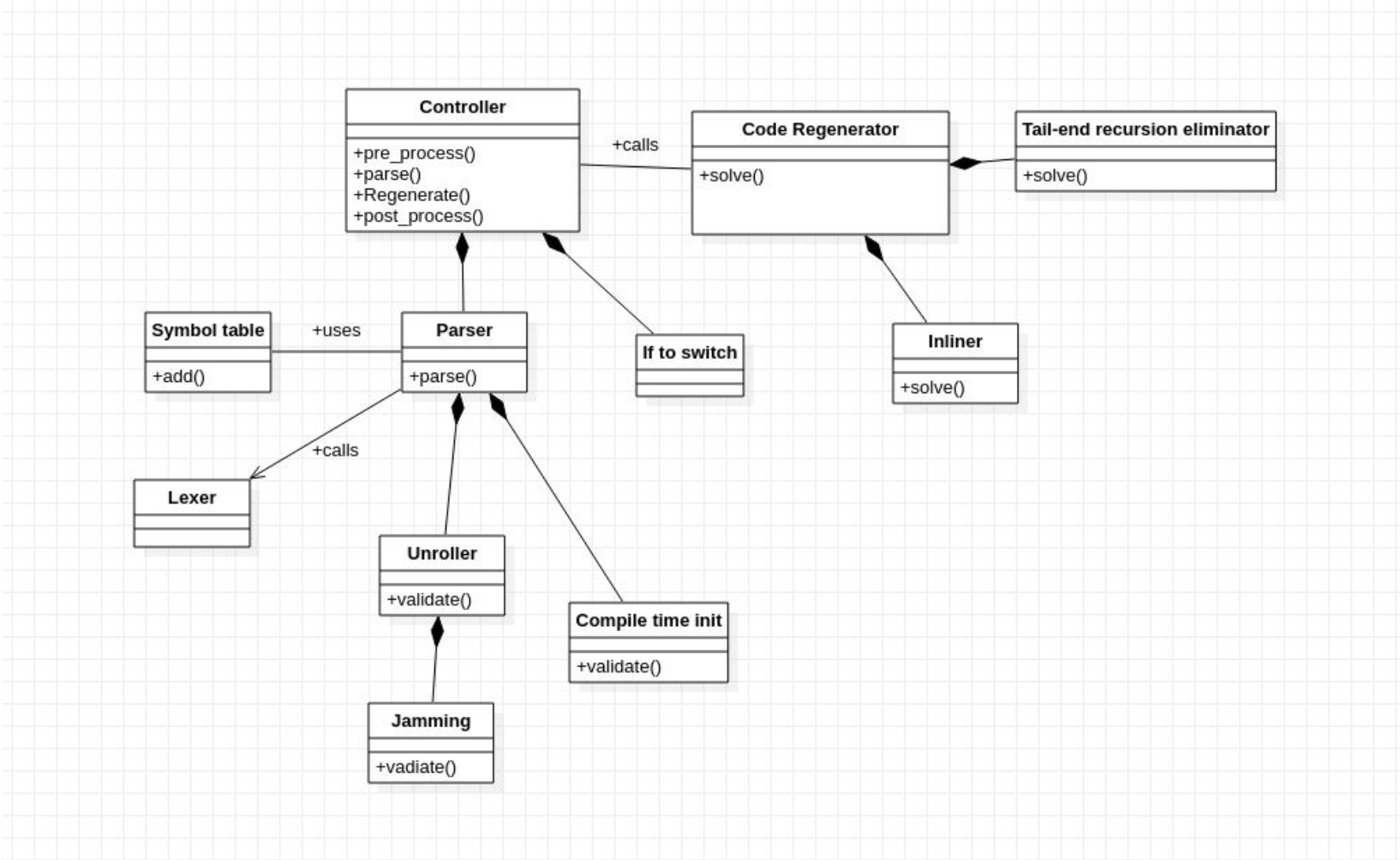
Algorithm:-

```
solve (start_index, end_index, PARSE_TREE, output_program) :
        if start_index=end_index // base case 1
                stop recursion
        if type (PARSE_TREE[start_index]) = 'string'  // base case 2
                if (PARSE_TREE[start_index] is a keyword)
                        output_program.append (PARSE_TREE[start_index] + space_char)
                else
                        output_program.append (PARSE_TREE[start_index])
        if type (PARSE_TREE[start_index]) = 'int'   // base case 3
                output_program.append (string(PARSE_TREE[start_index])
        if type (PARSE_TREE[start_index]) = 'list')
                solve (0, length(PARSE_TREE[start_index]), PARSE_TREE, output_program)
        solve (start_index+1, end_index, PARSE_TREE, output_program)  // continuing the
recursion
```

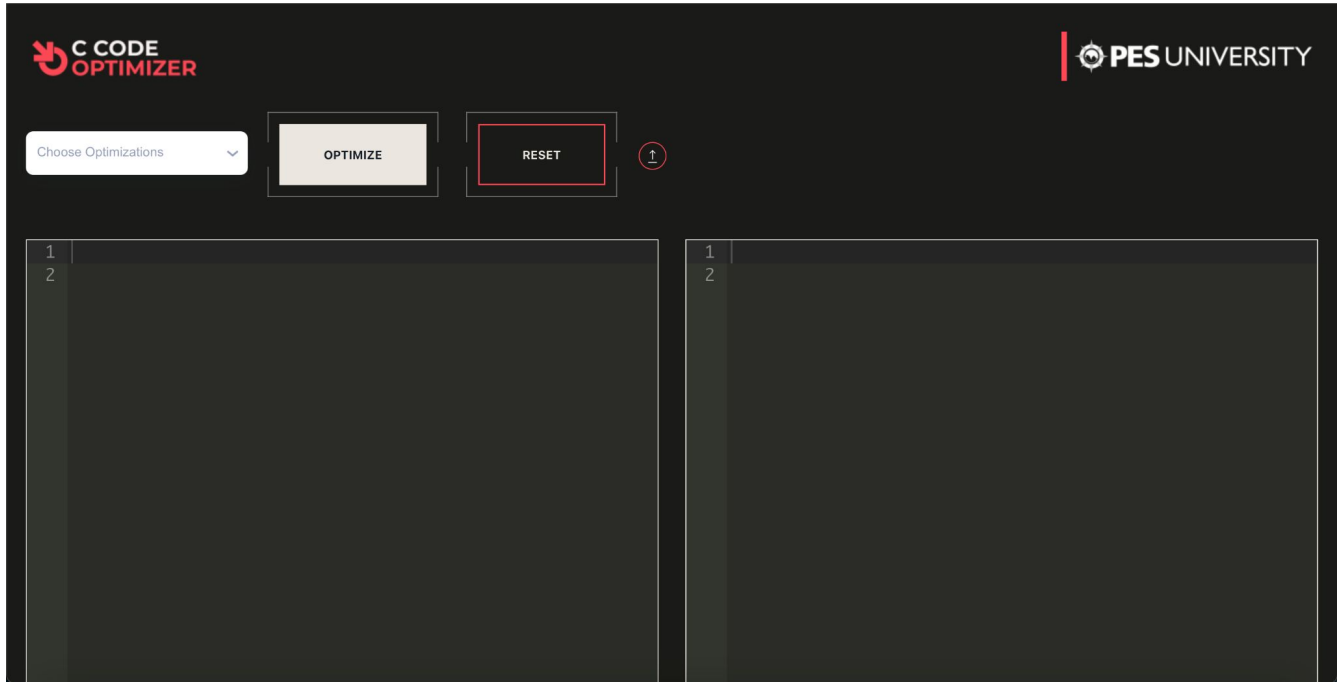
Architecture



Master Class Diagram



UI


C CODE
OPTIMIZER
PES UNIVERSITY
Choose Optimizations
OPTIMIZE
RESET

C CODE OPTIMIZER

PES UNIVERSITY

Tail Recursion Elimination ×

Function Inlining ×

OPTIMIZE

RESET

Download Code

```c
#include <stdio.h>

void sum(int size, double sum1)
{
    if (size == 0)
    {
        printf("sum : %f\n",sum1);
    }
    else sum(size - 1, sum1 + size);
}

int main()
{
    int size = 100000;
    sum(size, 0);
    return 0;
}
```

```c
// optimized code

#include<stdio.h>
void sum(int size, double sum1)
{
 label_1b728cc193614adf9e36b62180df9ee8:{
    }
    if (size == 0) { {
            printf("sum : %f\n", sum1);
    }
    } else { { {        // tail recursion eliminated
                int par_size_1b728cc193614adf9e36b62180df9
                    size;
                double par_sum1_1b728cc193614adf9e36b62180
                    = sum1;
                size =
                    par_size_1b728cc193614adf9e36b62180df9
                    1;
                sum1 =
```

OUTPUTS & METRICS

ubuntu@ubuntu: ~CCodeOptimizer (Non-Optimized Output)

File Edit View Search

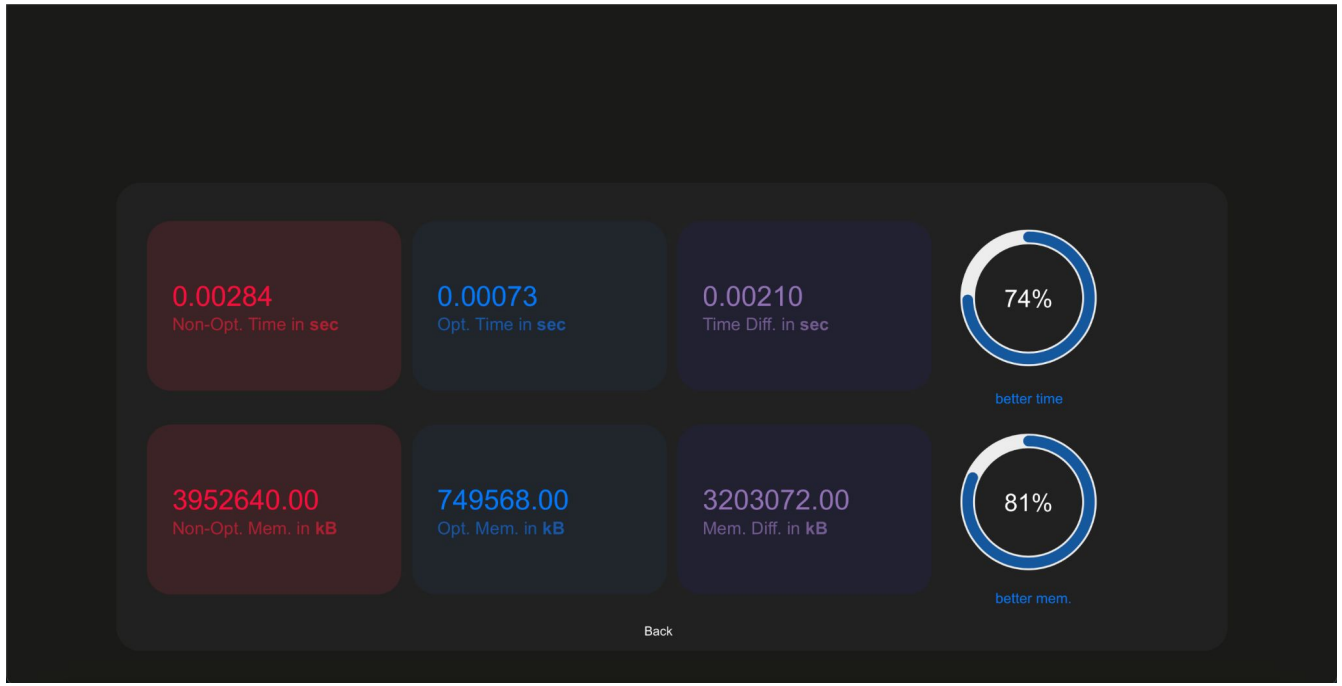
```
sum : 5000050000.000000
```

ubuntu@ubuntu: ~CCodeOptimizer (Optimized Output)

File Edit View Search

```
sum : 5000050000.000000
```

View Metrics


0.00284
Non-Opt. Time in sec
0.00073
Opt. Time in sec
0.00210
Time Diff. in sec
74%
better time
3952640.00
Non-Opt. Mem. in kB
749568.00
Opt. Mem. in kB
3203072.00
Mem. Diff. in kB
81%
better mem.
Back

Technologies Used

- ply3.11
- clang-tidy LLVM10.0.0
- gcc
- python3.6 or higher
- indent
- valgrind and kcachegrind
- gcc compiler
- Javascript
- PHP

Thank
You