Analyzing Computer System Configurations Using Valgrind

Selma Karasoftić and Amina Brković

January 12, 2025

Abstract

In this project, we analyze three distinct computer configurations designed for different use cases: gaming, word processing, and data processing. We used three different computers to evaluate performance, running a custom game we created, AbiWord for word processing, and GNU Octave for data processing. Using Valgrind for performance profiling, we measured the efficiency of each configuration under these specific workloads. Our analysis allowed us to identify the strengths and weaknesses of each setup, and based on our research, we provide tailored recommendations for the optimal system setup for gaming, word processing, and data processing tasks.

Contents

1 Introduction

This project investigates the performance of three distinct computer configurations, each performing specific tasks: gaming, word processing, and data processing. The primary goal is to show how well each configuration performs under realistic workloads and identify areas for optimization. To achieve this, we used Valgrind's profiling tools: Memcheck, to analyze memory usage and detect leaks or errors, and Cachegrind, for testing CPU and cache performance. These tools gave us insights into how efficiently the systems handled different applications.

The programs selected for evaluation represent real-world use cases. A lightweight terminal-based game was used to simulate gaming tasks, AbiWord served as the word processing application, and GNU Octave was employed for data-heavy computations. By running these programs on each configuration, we aimed to uncover strengths, weaknesses, and potential areas for improvement in memory and CPU performance.

Our research does not stop at providing performance insights; we have included logs and outputs for all the tests we did. These logs provide detailed evidence of the issues that were identified and the strengths observed, giving us a better analysis of the configurations. Some of our analysis was concluded on the full log (memcheck or cachegrind) which we will include in our submission. We did not include the screenshots of entire logs because some of them were too long, but the analysis was conducted considering the long log.

The findings and recommendations presented in this submission are intended to guide the optimization of these configurations, making them better suited for their intended tasks. By addressing identified inefficiencies and leveraging the strengths of each configuration, users can achieve higher performance, stability, and efficiency in their specific applications.

2 System Configurations

In this section, we provided detailed specifications of the three chosen computer configurations.

2.1 Gaming Configuration

- **CPU:** Intel Core i5-10300H 2.5 - 4.5 GHz
- **Memory:** 8GB DDR4 RAM
- **Cache:** 8MB L3 Cache
- **Graphics Card:** NVIDIA GeForce GTX 1650

2.2 Word Processing Configuration

- **CPU:** Intel Core i3-3217U 2 CPU cores running at 1.8- GHz
- **Memory:** 6GB DDR4 RAM
- **Cache:** 3MB L3 Cache

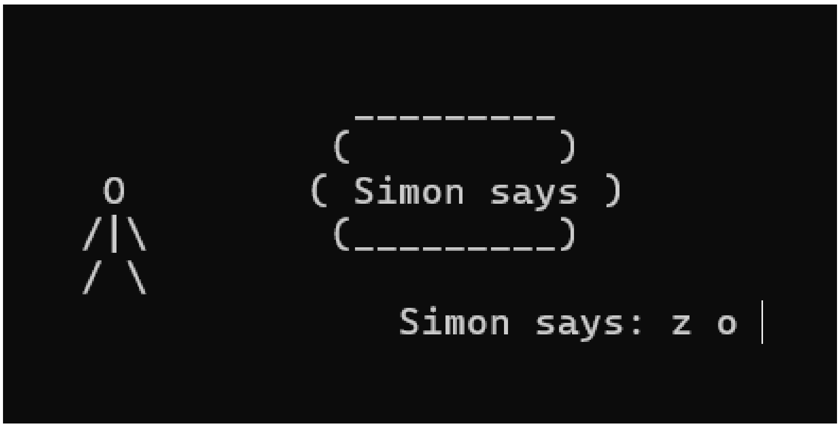
2.3 Data Processing Configuration

- **CPU:** AMD Ryzen 7 6800H with Radeon Graphics 3.2 GHz
- **Memory:** 16 GB DDR5-4800 MHz RAM.
- **Cache:** 16MB L3 Cache

3 Use Case Scenarios

Each configuration was evaluated by running a representative program for its respective use case:

3.1 Gaming



We created the game - **Simon Says: Memory Challenge,** that is a lightweight terminal-based game designed to test memory and attention span. The use case scenario involves using the CPU for real-time input handling, sequence generation, and display, which makes it suitable for testing efficient CPU memory access and moderate computational tasks. Although lightweight, the game uses simple terminal-based graphics (e.g., drawing the character and cloud) to test how the CPU handles frequent screen updates and character rendering efficiently.

- The game heavily relies on **CPU cycles** for random sequence generation (`rand()`), input processing, and rendering in the terminal using escape sequences (`gotoxy` and `clear_screen`).
- Real-time processing (via `kbhit()` and `getch()`) ensures that the system is stressed, simulating CPU-intensive scenarios.

The code for the game:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <termios.h>
#include <time.h>


void reset_terminal_mode();
void set_terminal_mode();
int kbhit();
char getch();
void clear_screen();
void gotoxy(int x, int y);
void draw_guy_and_cloud(int score);
void generate_sequence(char *sequence, int length);
void display_sequence(const char *sequence, int length);
int get_user_input(const char *sequence, int length);
void game_over(int score);

struct termios orig_termios;

void reset_terminal_mode() {
    tcsetattr(STDIN_FILENO, TCSANOW, &orig_termios);
}

void set_terminal_mode() {
    struct termios new_termios;
    tcgetattr(STDIN_FILENO, &orig_termios);
    atexit(reset_terminal_mode);
    new_termios = orig_termios;
```

```
    new_termios.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &new_termios);
}

int kbhit() {
    struct timeval tv = {0L, 0L};
    fd_set fds;
    FD_ZERO(&fds);
    FD_SET(STDIN_FILENO, &fds);
    return select(STDIN_FILENO + 1, &fds, NULL, NULL, &tv);
}

char getch() {
    char c;
    if (read(STDIN_FILENO, &c, 1) == 1) {
        return c;
    }
    return '\0';
}

void clear_screen() {
    printf("\033[2J");
    fflush(stdout);
}


void gotoxy(int x, int y) {
    printf("\033[%d;%dH", y + 1, x + 1);
    fflush(stdout);
}


void draw_guy_and_cloud(int score) {

    gotoxy(10, 10);
    printf("  O  ");
    gotoxy(10, 11);
    printf(" /|\\ ");
    gotoxy(10, 12);
    printf(" / \\ ");


    gotoxy(20, 8);
    printf("    _________");
    gotoxy(20, 9);
    printf("  (         )");
    gotoxy(20, 10);
    printf(" ( Simon says )");
    gotoxy(20, 11);
```

```
    printf("   (_________)");


    gotoxy(0, 0);
    printf("Current Score: %d", score);
    fflush(stdout);
}


void generate_sequence(char *sequence, int length) {
    const char commands[] = {'x', 'o', 'y', 'z'};
    for (int i = 0; i < length; i++) {
        sequence[i] = commands[rand() % 4];
    }
}


void display_sequence(const char *sequence, int length) {
    gotoxy(25, 13);
    printf("Simon says: ");
    fflush(stdout);
    usleep(500000);

    for (int i = 0; i < length; i++) {
        printf("%c ", sequence[i]);
        fflush(stdout);
        usleep(800000);
    }
    usleep(500000);
    gotoxy(25, 13);
    printf("                              ");
    fflush(stdout);
}


int get_user_input(const char *sequence, int length) {
    char input;
    gotoxy(25, 13);
    printf("Your turn (Press Enter to exit): ");
    fflush(stdout);

    for (int i = 0; i < length; i++) {
        input = getch();


        if (input == '\n') {
            return -1;
        }

```

```
        printf("%c ", input);
        fflush(stdout);


        if (input != sequence[i]) {
            return 0;
        }
    }
    return 1;
}


void game_over(int score) {
    gotoxy(10, 15);
    printf("Game Over!");
    gotoxy(10, 16);
    printf("Final Score: %d", score);
    gotoxy(10, 17);
    printf("Press any key to exit...");
    fflush(stdout);


    getch();
}

int main() {
    srand(time(0));
    set_terminal_mode();

    int score = 0;
    int sequence_length = 4;
    char sequence[8];

    while (1) {
        clear_screen();
        draw_guy_and_cloud(score);


        generate_sequence(sequence, sequence_length);
        display_sequence(sequence, sequence_length);


        int result = get_user_input(sequence, sequence_length);


        if (result == -1) {
            clear_screen();
            game_over(score);
            break;
```

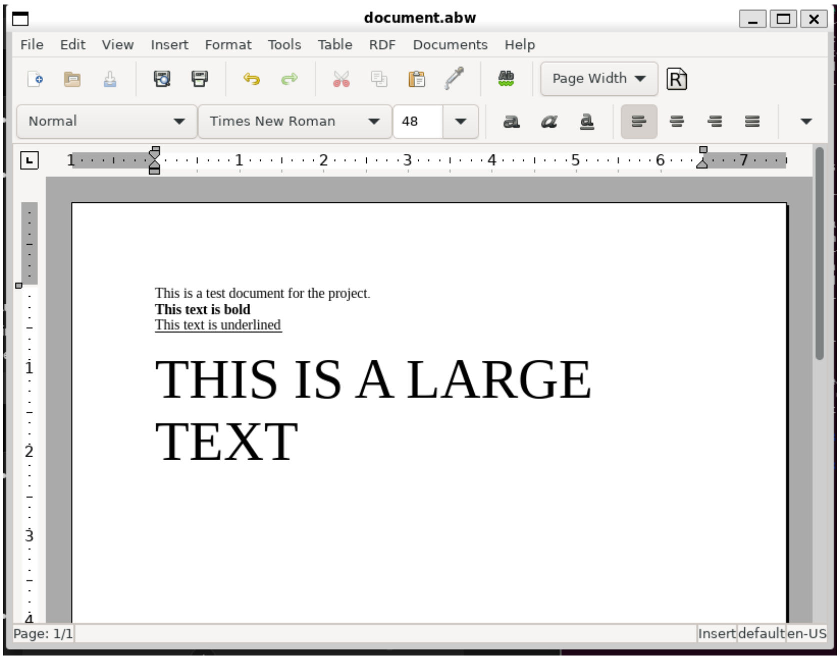
```
        }


        if (result == 0) {
            clear_screen();
            game_over(score);
            break;
        }


        score += sequence_length;
        if (sequence_length < 8) {
            sequence_length++;
        }
    }

    reset_terminal_mode();
    return 0;
}
```

3.2 Word Processing



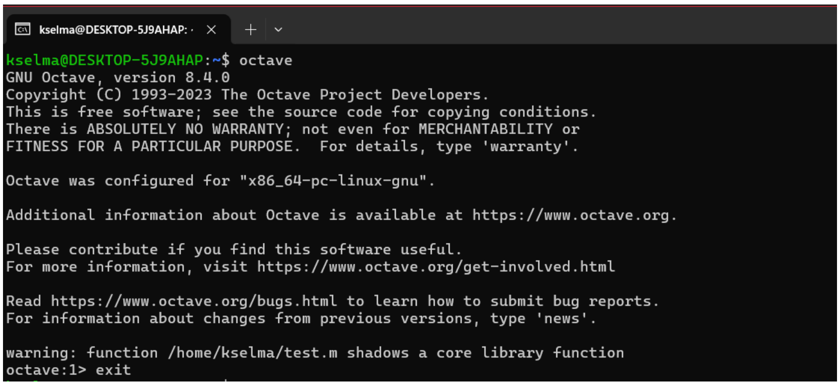
For word processing, we ran **AbiWord**, a lightweight word processor, to simulate typical office work. The test focused on evaluating CPU efficiency and moderate memory usage during common word processing tasks such as text editing, formatting, and document navigation. We chose AbiWord because it has a smaller memory footprint and faster startup times compared to LibreOffice Writer. In this case, we did the following things:

1. **Opening and editing a document** to evaluate memory allocation and management during text operations.

2. **Applying basic formatting (e.g., bold, underlined, and text size)** to simulate typical office workloads.

This setup provided insights into how the CPU handled lightweight word processing tasks and how memory was utilized efficiently in real-time scenarios, ensuring a smooth and responsive experience for the user.

3.3 Data Processing



A GNU Octave script was used to perform matrix operations, simulating data-heavy computational tasks, and testing the CPU's ability to handle intensive workloads. The script leveraged Octave's efficient matrix manipulation capabilities to generate a random 100x100 matrix and perform various computations, including basic statistics, row and column operations, element-wise transformations, matrix normalization, variance and standard deviation calculations, and sorting. This approach utilized Octave's inherent matrix-centric design to efficiently process and analyze the data, providing a robust test of computational performance. The code that we used for this use case:

```
matrix = rand(100, 100);

disp("Calculating basic statistics...");
mean_value = mean(matrix(:));
max_value = max(matrix(:));
min_value = min(matrix(:));
disp(["Mean value: ", num2str(mean_value)]);
disp(["Max value: ", num2str(max_value)]);
disp(["Min value: ", num2str(min_value)]);

disp("Calculating row and column sums...");
row_sums = sum(matrix, 2);
column_sums = sum(matrix, 1);
disp("First 5 row sums:");
disp(row_sums(1:5));
disp("First 5 column sums:");
disp(column_sums(1:5));

disp("Applying element-wise transformation...");
transformed_matrix = log(matrix + 1);
disp("Displaying a portion of the transformed matrix:");
disp(transformed_matrix(1:5, 1:5));
```

```

disp("Normalizing the matrix...");
normalized_matrix = (matrix - min_value) / (max_value - min_value);
disp("Displaying a portion of the normalized matrix:");
disp(normalized_matrix(1:5, 1:5));

disp("Calculating variance and standard deviation...");
variance = var(matrix(:));
std_dev = std(matrix(:));
disp(["Variance: ", num2str(variance)]);
disp(["Standard deviation: ", num2str(std_dev)]);

disp("Sorting matrix values...");
sorted_values = sort(matrix(:));
disp("First 5 sorted values:");
disp(sorted_values(1:5));
disp("Last 5 sorted values:");
disp(sorted_values(end-4:end));
```

4 Valgrind Profiling Results

For this part, we used memcheck and cachegrind tools of Valgrind. We tested all use cases on every device and recorded every log that we got. Memcheck is a tool within Valgrind designed to identify memory-related issues in C and C++ applications. It shows errors such as leaks, incorrect memory accesses, usage of uninitialized memory, redundant frees, and memory corruption. Running Memcheck on our code enables us to spot issues like buffer overflows, errors with handling of the dynamic memory, and problems with variables. Cachegrind is a tool within Valgrind is used for testing activities related to the CPU. It simulates the behavior of L1, L2, and other cache levels, giving us insights into cache hits, misses, and their respective rates. Additionally, Cachegrind tracks success and failure rates. Cachegrind helps us to enhance performance and remove bottlenecks in software applications with substantial memory dependencies. Cachegrind covers all the logs that callgrind does too so the final summary is the same for both.

4.1 Memcheck Results

Gaming Configuration:

Result for Game program

```
==1640== Memcheck, a memory error detector
==1640== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==1640== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==1640== Command: ./GameStart
==1640==
==1640==
==1640== HEAP SUMMARY:
==1640==     in use at exit: 0 bytes in 0 blocks
==1640==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==1640==
==1640== All heap blocks were freed -- no leaks are possible
==1640==
==1640== For lists of detected and suppressed errors, rerun with: -s
==1640== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Result for Word Processing program

```
==1754==    by 0x50C4BB0: wcsxfrm_l (strxfrm_l.c:679)
==1754==    by 0x6B151C4: g_utf8_collate_key (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6B15706: g_utf8_collate_key_for_filename (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x59AE44F: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==1754==    by 0x59BA357: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==1754==    by 0x5C47980: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==1754==    by 0x59B3AAC: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==1754==    by 0x6AE6B24: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6AE6865: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6AE6857: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6AE6857: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==  Address 0x10a4a850 is 0 bytes inside a block of size 16 alloc'd
==1754==    at 0x4846828: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==1754==    by 0x6ADDAF9: g_malloc (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6B14BF5: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6B15187: g_utf8_collate_key (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6B15706: g_utf8_collate_key_for_filename (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x59AE44F: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==1754==    by 0x59BA357: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==1754==    by 0x5C47980: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==1754==    by 0x59B3AAC: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==1754==    by 0x6AE6B24: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6AE6865: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6AE6857: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==
==1754== Invalid read of size 32
==1754==    at 0x517BB2E: __wcpncpy_avx2 (strncpy-avx2.S:85)
==1754==    by 0x50C4BB0: wcsxfrm_l (strxfrm_l.c:679)
==1754==    by 0x6B151C4: g_utf8_collate_key (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6B15449: g_utf8_collate_key_for_filename (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x59AE44F: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==1754==    by 0x59BA357: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==1754==    by 0x5C47980: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==1754==    by 0x59B3AAC: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==1754==    by 0x6AE6B24: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6AE6865: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6AE6865: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6AE6865: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==  Address 0x10df2350 is 0 bytes inside a block of size 28 alloc'd
==1754==    at 0x4846828: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==1754==    by 0x6ADDAF9: g_malloc (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6B14BF5: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6B15187: g_utf8_collate_key (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6B15449: g_utf8_collate_key_for_filename (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x59AE44F: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==1754==    by 0x59BA357: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==1754==    by 0x5C47980: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==1754==    by 0x59B3AAC: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==1754==    by 0x6AE6B24: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6AE6865: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==    by 0x6AE6865: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==1754==
==1754==
==1754== HEAP SUMMARY:
==1754==     in use at exit: 4,423,735 bytes in 44,957 blocks
==1754==   total heap usage: 1,080,228 allocs, 1,035,271 frees, 242,487,909 bytes allocated
==1754==
==1754== LEAK SUMMARY:
==1754==    definitely lost: 71,049 bytes in 198 blocks
==1754==    indirectly lost: 132,928 bytes in 4,942 blocks
==1754==      possibly lost: 12,384 bytes in 126 blocks
==1754==    still reachable: 3,954,094 bytes in 37,588 blocks
==1754==                       of which reachable via heuristic:
==1754==                         newarray           : 1,592 bytes in 1 blocks
==1754==         suppressed: 0 bytes in 0 blocks
==1754== Rerun with --leak-check=full to see details of leaked memory
==1754==
==1754== Use --track-origins=yes to see where uninitialised values come from
==1754== For lists of detected and suppressed errors, rerun with: -s
==1754== ERROR SUMMARY: 33 errors from 15 contexts (suppressed: 0 from 0)
```

Result for Data Processing program

```
kselma@DESKTOP-5J9AHAP: ~    kselma@DESKTOP-5J9AHAP: ·
Calculating basic statistics...
Mean value: 0.49684
Max value: 0.99997
Min value: 0.00015122
Calculating row and column sums...
First 5 row sums:
   48.055
   47.965
   47.775
   54.385
   51.598
First 5 column sums:
   50.218   52.306   50.301   52.143   43.534
Applying element-wise transformation...
Displaying a portion of the transformed matrix:
   0.385188   0.339701   0.340425   0.520368   0.010174
   0.671777   0.082201   0.455053   0.119065   0.118162
   0.436890   0.108643   0.475435   0.459738   0.518214
   0.565035   0.669361   0.038847   0.040240   0.114975
   0.374554   0.647841   0.533482   0.524372   0.631845
Normalizing the matrix...
Displaying a portion of the normalized matrix:
   0.469825   0.404449   0.405466   0.682619   0.010077
   0.957734   0.085539   0.576210   0.126314   0.125298
   0.547833   0.114634   0.608672   0.583613   0.678998
   0.759494   0.953010   0.039467   0.040917   0.121717
   0.454273   0.911423   0.704833   0.689370   0.881086
Calculating variance and standard deviation...
Variance: 0.08486
Standard deviation: 0.29131
Sorting matrix values...
First 5 sorted values:
   1.5122e-04
   1.6912e-04
   2.5803e-04
   2.6979e-04
   4.0647e-04
Last 5 sorted values:
   0.9998
   0.9998
   0.9999
   0.9999
   1.0000
octave:35> exit
==1953==
==1953== HEAP SUMMARY:
==1953==     in use at exit: 458,773 bytes in 4,123 blocks
==1953==   total heap usage: 326,715 allocs, 322,592 frees, 48,066,267 bytes allocated
==1953==
==1953== LEAK SUMMARY:
==1953==    definitely lost: 839 bytes in 61 blocks
==1953==    indirectly lost: 158,604 bytes in 2,880 blocks
==1953==      possibly lost: 480 bytes in 1 blocks
==1953==    still reachable: 298,850 bytes in 1,181 blocks
==1953==         suppressed: 0 bytes in 0 blocks
==1953== Rerun with --leak-check=full to see details of leaked memory
==1953==
==1953== For lists of detected and suppressed errors, rerun with: -s
==1953== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Word Processing Configuration:

Result for Game program

```
==1497== Memcheck, a memory error detector
==1497== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==1497== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==1497== Command: ./GameStart
==1497==
==1497==
==1497== HEAP SUMMARY:
==1497==     in use at exit: 0 bytes in 0 blocks
==1497==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==1497==
==1497== All heap blocks were freed -- no leaks are possible
==1497==
==1497== For lists of detected and suppressed errors, rerun with: -s
==1497== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Result for Word Processing program

```
==2006==    by 0x62573AF: ??? (in /usr/lib/x86_64-linux-gnu/librsvg-2.so.2.50.0)
==2006==    by 0x627DA50: ??? (in /usr/lib/x86_64-linux-gnu/librsvg-2.so.2.50.0)
==2006==    by 0x6282839: ??? (in /usr/lib/x86_64-linux-gnu/librsvg-2.so.2.50.0)
==2006==    by 0x6A58A7C: g_type_class_ref (in /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0.8000.0)
==2006==    by 0x6A40161: g_object_new_with_properties (in /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0.8000.0)
==2006==    by 0x659A2E8: ??? (in /usr/lib/x86_64-linux-gnu/librsvg-2.so.2.50.0)
==2006==    by 0x627499C: rsvg_handle_new (in /usr/lib/x86_64-linux-gnu/librsvg-2.so.2.50.0)
==2006==    by 0x11C0C3DB: ??? (in /usr/lib/x86_64-linux-gnu/gdk-pixbuf-2.0/2.10.0/loaders/libpixbufloader-svg.so)
==2006==    by 0x69F871C: ??? (in /usr/lib/x86_64-linux-gnu/libgdk_pixbuf-2.0.so.0.4200.10)
==2006==    by 0x69FD2C5: gdk_pixbuf_loader_close (in /usr/lib/x86_64-linux-gnu/libgdk_pixbuf-2.0.so.0.4200.10)
==2006==
==2006== Conditional jump or move depends on uninitialised value(s)
==2006==    at 0x6598346: ??? (in /usr/lib/x86_64-linux-gnu/librsvg-2.so.2.50.0)
==2006==    by 0x6257DC9: ??? (in /usr/lib/x86_64-linux-gnu/librsvg-2.so.2.50.0)
==2006==    by 0x62573AF: ??? (in /usr/lib/x86_64-linux-gnu/librsvg-2.so.2.50.0)
==2006==    by 0x627DA50: ??? (in /usr/lib/x86_64-linux-gnu/librsvg-2.so.2.50.0)
==2006==    by 0x6282839: ??? (in /usr/lib/x86_64-linux-gnu/librsvg-2.so.2.50.0)
==2006==    by 0x6A58A7C: g_type_class_ref (in /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0.8000.0)
==2006==    by 0x6A40161: g_object_new_with_properties (in /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0.8000.0)
==2006==    by 0x659A2E8: ??? (in /usr/lib/x86_64-linux-gnu/librsvg-2.so.2.50.0)
==2006==    by 0x627499C: rsvg_handle_new (in /usr/lib/x86_64-linux-gnu/librsvg-2.so.2.50.0)
==2006==    by 0x11C0C3DB: ??? (in /usr/lib/x86_64-linux-gnu/gdk-pixbuf-2.0/2.10.0/loaders/libpixbufloader-svg.so)
==2006==    by 0x69F871C: ??? (in /usr/lib/x86_64-linux-gnu/libgdk_pixbuf-2.0.so.0.4200.10)
==2006==    by 0x69FD2C5: gdk_pixbuf_loader_close (in /usr/lib/x86_64-linux-gnu/libgdk_pixbuf-2.0.so.0.4200.10)
==2006==
==2006==
==2006== HEAP SUMMARY:
==2006==     in use at exit: 4,430,473 bytes in 44,698 blocks
==2006==   total heap usage: 1,253,183 allocs, 1,208,485 frees, 212,729,638 bytes allocated
==2006==
==2006== LEAK SUMMARY:
==2006==    definitely lost: 71,920 bytes in 193 blocks
==2006==    indirectly lost: 131,728 bytes in 4,956 blocks
==2006==      possibly lost: 12,416 bytes in 126 blocks
==2006==    still reachable: 3,960,369 bytes in 37,320 blocks
==2006==                       of which reachable via heuristic:
==2006==                         newarray           : 1,592 bytes in 1 blocks
==2006==         suppressed: 0 bytes in 0 blocks
==2006== Rerun with --leak-check=full to see details of leaked memory
==2006==
==2006== Use --track-origins=yes to see where uninitialised values come from
==2006== For lists of detected and suppressed errors, rerun with: -s
==2006== ERROR SUMMARY: 12 errors from 12 contexts (suppressed: 0 from 0)
```

Result for Data Processing program

```
kselma@DESKTOP-J834KO6: ~
   8.7735e-02   2.6936e-01   6.1219e-01   4.9077e-01   4.0097e-01
   5.9382e-01   1.2182e-01   2.3961e-01   5.6800e-01   1.6731e-01
   4.7138e-01   2.2840e-01   6.0209e-01   5.3499e-01   3.1803e-01
   6.7465e-01   4.9728e-01   1.9947e-01   2.9243e-01   2.7567e-01
   4.8918e-01   8.0010e-03   2.2237e-01   7.9348e-02   6.5033e-01
Normalizing the matrix...
Displaying a portion of the normalized matrix:
   9.1664e-02   3.0911e-01   8.4448e-01   6.3358e-01   4.9327e-01
   8.1090e-01   1.2951e-01   2.7073e-01   7.6475e-01   1.8209e-01
   6.0220e-01   2.5657e-01   8.2595e-01   7.0744e-01   3.7440e-01
   9.6337e-01   6.4425e-01   2.2073e-01   3.3966e-01   3.1739e-01
   6.3097e-01   7.9926e-03   2.4901e-01   8.2545e-02   9.1618e-01
Calculating variance and standard deviation...
Variance: 0.082932
Standard deviation: 0.28798
Sorting matrix values...
First 5 sorted values:
   4.1055e-05
   2.2403e-04
   3.1928e-04
   3.5172e-04
   5.4488e-04
Last 5 sorted values:
   0.9995
   0.9997
   0.9999
   1.0000
   1.0000
==2229==
==2229== HEAP SUMMARY:
==2229==     in use at exit: 190,755 bytes in 3,199 blocks
==2229==   total heap usage: 301,458 allocs, 298,259 frees, 42,838,382 bytes allocated
==2229==
==2229== LEAK SUMMARY:
==2229==    definitely lost: 839 bytes in 61 blocks
==2229==    indirectly lost: 158,604 bytes in 2,880 blocks
==2229==      possibly lost: 480 bytes in 1 blocks
==2229==    still reachable: 30,832 bytes in 257 blocks
==2229==         suppressed: 0 bytes in 0 blocks
==2229== Rerun with --leak-check=full to see details of leaked memory
==2229==
==2229== For lists of detected and suppressed errors, rerun with: -s
==2229== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Data Processing Configuration:

Result for Game program

```
==8068== Memcheck, a memory error detector
==8068== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==8068== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==8068== Command: ./GameStart
==8068==
==8068==
==8068== HEAP SUMMARY:
==8068==     in use at exit: 0 bytes in 0 blocks
==8068==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==8068==
==8068== All heap blocks were freed -- no leaks are possible
==8068==
==8068== For lists of detected and suppressed errors, rerun with: -s
==8068== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Result for Word Processing program

```
==8258==    by 0x6B1568D: g_utf8_collate_key_for_filename (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==8258==    by 0x59AE44F: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==8258==    by 0x59BA357: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==8258==    by 0x5C4799B: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==8258==    by 0x59B3AAC: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==8258==    by 0x6AE6B24: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==8258==    by 0x6AE6857: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==8258==    by 0x6AE6865: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==8258==    by 0x6AE6857: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==8258==  Address 0x147a63f0 is 0 bytes inside a block of size 16 alloc'd
==8258==    at 0x4846828: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==8258==    by 0x6ADDAF9: g_malloc (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==8258==    by 0x6B14BF5: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==8258==    by 0x6B15187: g_utf8_collate_key (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==8258==    by 0x6B1568D: g_utf8_collate_key_for_filename (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==8258==    by 0x59AE44F: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==8258==    by 0x59BA357: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==8258==    by 0x5C4799B: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==8258==    by 0x59B3AAC: ??? (in /usr/lib/x86_64-linux-gnu/libgtk-3.so.0.2409.32)
==8258==    by 0x6AE6B24: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==8258==    by 0x6AE6857: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==8258==    by 0x6AE6865: ??? (in /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0.8000.0)
==8258==
==8258==
==8258== HEAP SUMMARY:
==8258==     in use at exit: 4,488,268 bytes in 45,606 blocks
==8258==   total heap usage: 1,253,367 allocs, 1,207,761 frees, 202,316,639 bytes allocated
==8258==
==8258== LEAK SUMMARY:
==8258==    definitely lost: 71,488 bytes in 196 blocks
==8258==    indirectly lost: 139,360 bytes in 4,959 blocks
==8258==      possibly lost: 12,416 bytes in 126 blocks
==8258==    still reachable: 4,011,028 bytes in 38,223 blocks
==8258==                       of which reachable via heuristic:
==8258==                         newarray           : 1,592 bytes in 1 blocks
==8258==         suppressed: 0 bytes in 0 blocks
==8258== Rerun with --leak-check=full to see details of leaked memory
==8258==
==8258== Use --track-origins=yes to see where uninitialised values come from
==8258== For lists of detected and suppressed errors, rerun with: -s
==8258== ERROR SUMMARY: 17 errors from 13 contexts (suppressed: 0 from 0)
```

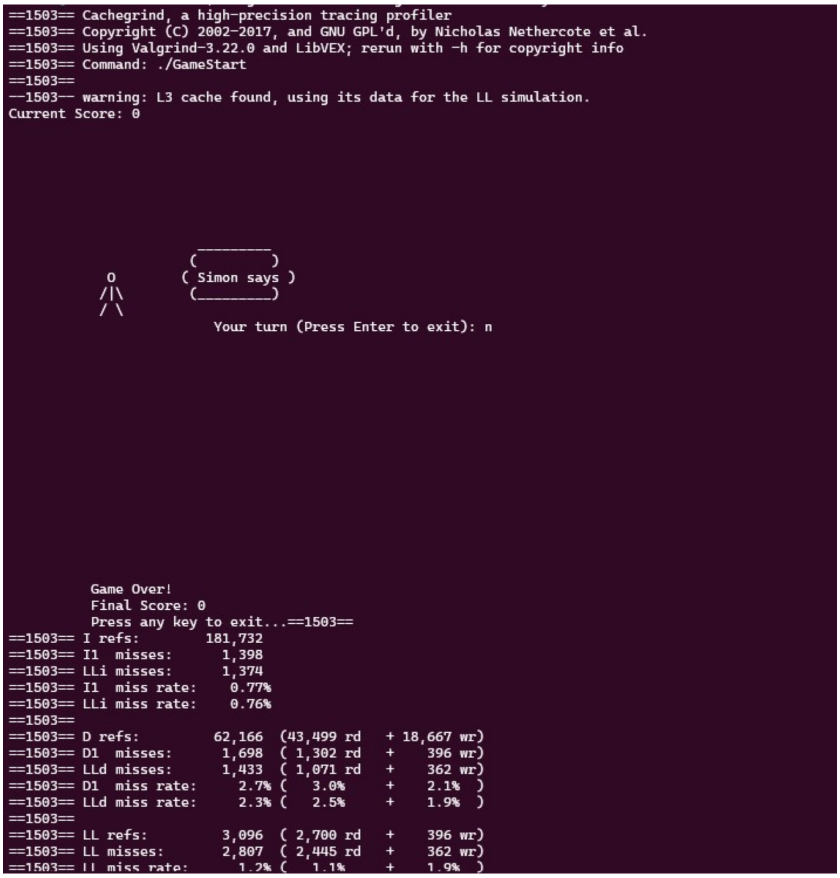
Result for Data Processing program

```
Calculating row and column sums...
First 5 row sums:
   46.343
   50.347
   56.945
   45.943
   49.641
First 5 column sums:
   50.950   46.109   52.099   52.716   52.610
Applying element-wise transformation...
Displaying a portion of the transformed matrix:
   0.352059   0.507395   0.615543   0.136566   0.035760
   0.493648   0.076334   0.642384   0.673261   0.564248
   0.595425   0.663748   0.278873   0.529041   0.103246
   0.672899   0.557408   0.347422   0.231227   0.524604
   0.562001   0.416429   0.635313   0.505754   0.237828
Normalizing the matrix...
Displaying a portion of the normalized matrix:
   0.422057   0.661075   0.850818   0.146336   0.036389
   0.638394   0.079314   0.901176   0.960802   0.758262
   0.813951   0.942234   0.321683   0.697428   0.108761
   0.960091   0.746275   0.415477   0.260175   0.689913
   0.754316   0.516621   0.887777   0.658351   0.268522
Calculating variance and standard deviation...
Variance: 0.083019
Standard deviation: 0.28813
Sorting matrix values...
First 5 sorted values:
   2.6081e-05
   1.0630e-04
   2.6092e-04
   3.0305e-04
   3.8426e-04
Last 5 sorted values:
   0.9995
   0.9996
   0.9997
   0.9998
   0.9998
octave:35> exit
==8757==
==8757== HEAP SUMMARY:
==8757==     in use at exit: 423,719 bytes in 3,386 blocks
==8757==   total heap usage: 325,303 allocs, 321,917 frees, 47,967,732 bytes allocated
==8757==
==8757== LEAK SUMMARY:
==8757==    definitely lost: 839 bytes in 61 blocks
==8757==    indirectly lost: 158,604 bytes in 2,880 blocks
==8757==      possibly lost: 480 bytes in 1 blocks
==8757==    still reachable: 263,796 bytes in 444 blocks
==8757==         suppressed: 0 bytes in 0 blocks
==8757== Rerun with --leak-check=full to see details of leaked memory
==8757==
==8757== For lists of detected and suppressed errors, rerun with: -s
==8757== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
brki@DESKTOP-5REVLB6:~$ |
```

4.2 Callgrind Results

Gaming Configuration:

Result for Game program

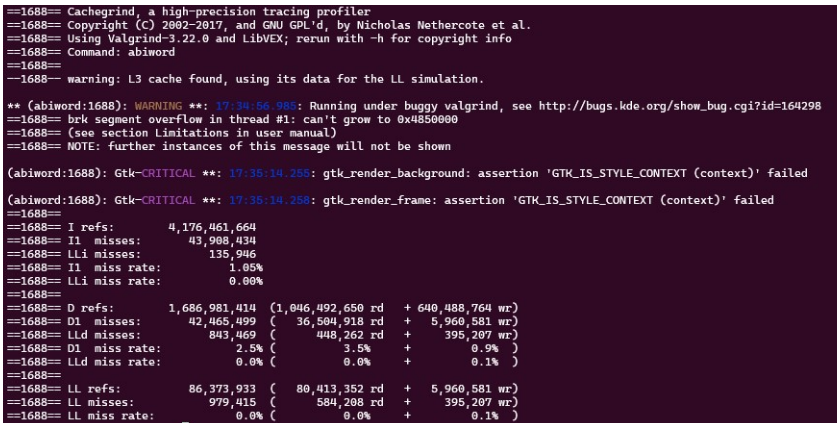


```
==1503== Cachegrind, a high-precision tracing profiler
==1503== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==1503== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==1503== Command: ./GameStart
==1503==
--1503-- warning: L3 cache found, using its data for the LL simulation.
Current Score: 0

                         _________
                        (         )
             O         ( Simon says )
            /|\         (_________)
            / \
                          Your turn (Press Enter to exit): n

           Game Over!
           Final Score: 0
           Press any key to exit...==1503==
==1503== I refs:        181,732
==1503== I1  misses:      1,398
==1503== LLi misses:      1,374
==1503== I1  miss rate:    0.77%
==1503== LLi miss rate:    0.76%
==1503==
==1503== D refs:         62,166  (43,499 rd   + 18,667 wr)
==1503== D1  misses:      1,698  ( 1,302 rd   +    396 wr)
==1503== LLd misses:      1,433  ( 1,071 rd   +    362 wr)
==1503== D1  miss rate:     2.7% (   3.0%     +    2.1%  )
==1503== LLd miss rate:     2.3% (   2.5%     +    1.9%  )
==1503==
==1503== LL refs:         3,096  ( 2,700 rd   +    396 wr)
==1503== LL misses:       2,807  ( 2,445 rd   +    362 wr)
==1503== LL miss rate:      1.2% (   1.1%     +    1.9%  )
```

Result for Word Processing program



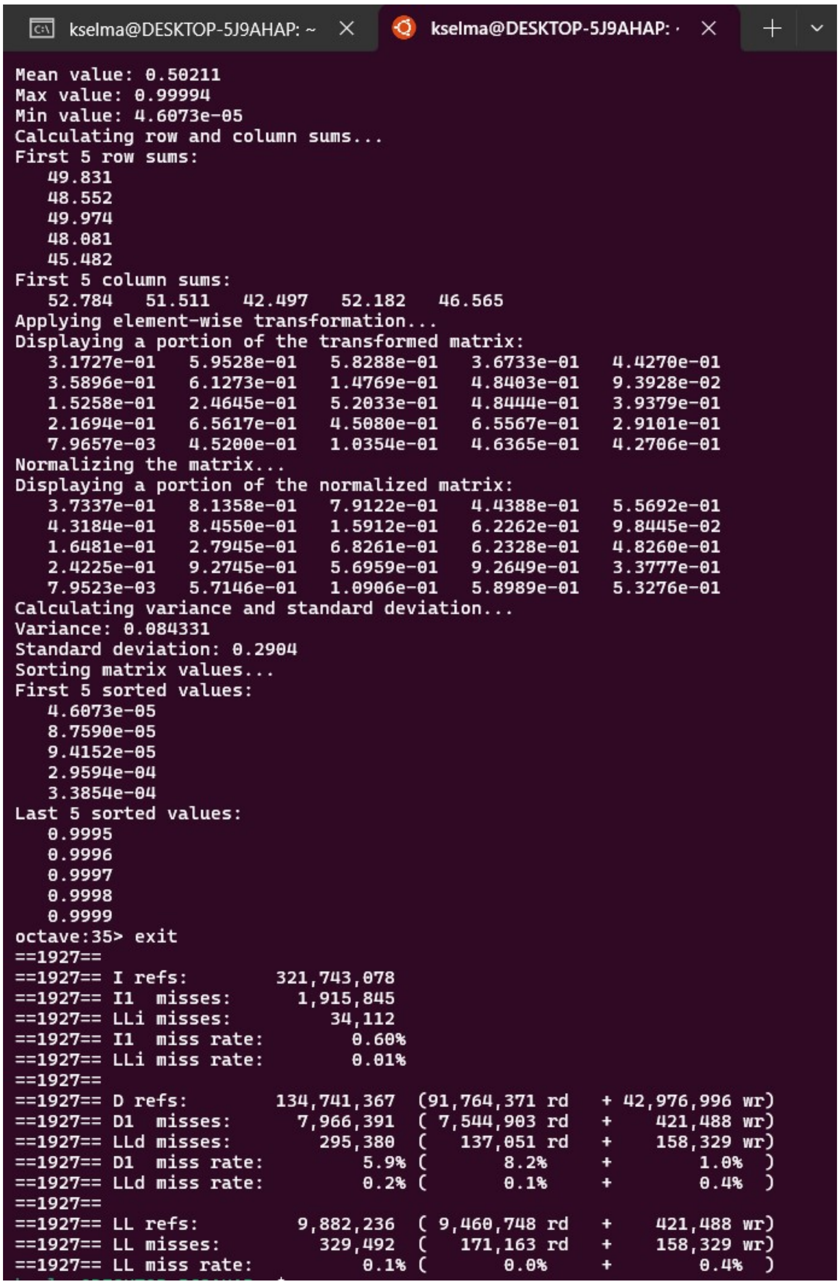
```
==1688== Cachegrind, a high-precision tracing profiler
==1688== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==1688== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==1688== Command: abiword
==1688==
--1688-- warning: L3 cache found, using its data for the LL simulation.

** (abiword:1688): WARNING **: 17:34:56.985: Running under buggy valgrind, see http://bugs.kde.org/show_bug.cgi?id=164298
==1688== brk segment overflow in thread #1: can't grow to 0x4850000
==1688== (see section Limitations in user manual)
==1688== NOTE: further instances of this message will not be shown

(abiword:1688): Gtk-CRITICAL **: 17:35:14.255: gtk_render_background: assertion 'GTK_IS_STYLE_CONTEXT (context)' failed

(abiword:1688): Gtk-CRITICAL **: 17:35:14.258: gtk_render_frame: assertion 'GTK_IS_STYLE_CONTEXT (context)' failed
==1688==
==1688== I refs:        4,176,461,664
==1688== I1  misses:       43,908,434
==1688== LLi misses:          135,946
==1688== I1  miss rate:          1.05%
==1688== LLi miss rate:          0.00%
==1688==
==1688== D refs:        1,686,981,414  (1,046,492,650 rd   + 640,488,764 wr)
==1688== D1  misses:       42,465,499  (   36,504,918 rd   +   5,960,581 wr)
==1688== LLd misses:          843,469  (      448,262 rd   +     395,207 wr)
==1688== D1  miss rate:           2.5% (          3.5%     +         0.9%  )
==1688== LLd miss rate:           0.0% (          0.0%     +         0.1%  )
==1688==
==1688== LL refs:          86,373,933  (   80,413,352 rd   +   5,960,581 wr)
==1688== LL misses:           979,415  (      584,208 rd   +     395,207 wr)
==1688== LL miss rate:            0.0% (          0.0%     +         0.1%  )
```

Result for Data Processing program



```
Mean value: 0.50211
Max value: 0.99994
Min value: 4.6073e-05
Calculating row and column sums...
First 5 row sums:
   49.831
   48.552
   49.974
   48.081
   45.482
First 5 column sums:
   52.784   51.511   42.497   52.182   46.565
Applying element-wise transformation...
Displaying a portion of the transformed matrix:
   3.1727e-01   5.9528e-01   5.8288e-01   3.6733e-01   4.4270e-01
   3.5896e-01   6.1273e-01   1.4769e-01   4.8403e-01   9.3928e-02
   1.5258e-01   2.4645e-01   5.2033e-01   4.8444e-01   3.9379e-01
   2.1694e-01   6.5617e-01   4.5080e-01   6.5567e-01   2.9101e-01
   7.9657e-03   4.5200e-01   1.0354e-01   4.6365e-01   4.2706e-01
Normalizing the matrix...
Displaying a portion of the normalized matrix:
   3.7337e-01   8.1358e-01   7.9122e-01   4.4388e-01   5.5692e-01
   4.3184e-01   8.4550e-01   1.5912e-01   6.2262e-01   9.8445e-02
   1.6481e-01   2.7945e-01   6.8261e-01   6.2328e-01   4.8260e-01
   2.4225e-01   9.2745e-01   5.6959e-01   9.2649e-01   3.3777e-01
   7.9523e-03   5.7146e-01   1.0906e-01   5.8989e-01   5.3276e-01
Calculating variance and standard deviation...
Variance: 0.084331
Standard deviation: 0.2904
Sorting matrix values...
First 5 sorted values:
   4.6073e-05
   8.7590e-05
   9.4152e-05
   2.9594e-04
   3.3854e-04
Last 5 sorted values:
   0.9995
   0.9996
   0.9997
   0.9998
   0.9999
octave:35> exit
==1927==
==1927== I refs:        321,743,078
==1927== I1  misses:      1,915,845
==1927== LLi misses:         34,112
==1927== I1  miss rate:        0.60%
==1927== LLi miss rate:        0.01%
==1927==
==1927== D refs:        134,741,367  (91,764,371 rd   + 42,976,996 wr)
==1927== D1  misses:      7,966,391  ( 7,544,903 rd   +    421,488 wr)
==1927== LLd misses:        295,380  (   137,051 rd   +    158,329 wr)
==1927== D1  miss rate:         5.9% (       8.2%     +        1.0%  )
==1927== LLd miss rate:         0.2% (       0.1%     +        0.4%  )
==1927==
==1927== LL refs:         9,882,236  ( 9,460,748 rd   +    421,488 wr)
==1927== LL misses:         329,492  (   171,163 rd   +    158,329 wr)
==1927== LL miss rate:          0.1% (       0.0%     +        0.4%  )
```

Word Processing Configuration:

Result for Game program

```
                 ________
                (        )
      O        ( Simon says )
     /|\        (________)
     / \
                     Your turn (Press Enter to exit): m











         Game Over!
         Final Score: 0
         Press any key to exit...==1336==
==1336== I refs:        179,789
==1336== I1  misses:      1,379
==1336== LLi misses:      1,353
==1336== I1  miss rate:    0.77%
==1336== LLi miss rate:    0.75%
==1336==
==1336== D refs:         61,811  (43,206 rd   + 18,605 wr)
==1336== D1  misses:      1,700  ( 1,305 rd   +    395 wr)
==1336== LLd misses:      1,431  ( 1,066 rd   +    365 wr)
==1336== D1  miss rate:     2.8% (   3.0%     +    2.1%  )
==1336== LLd miss rate:     2.3% (   2.5%     +    2.0%  )
==1336==
==1336== LL refs:         3,079  ( 2,684 rd   +    395 wr)
==1336== LL misses:       2,784  ( 2,419 rd   +    365 wr)
==1336== LL miss rate:      1.2% (   1.1%     +    2.0%  )
```

Result for Word Processing program

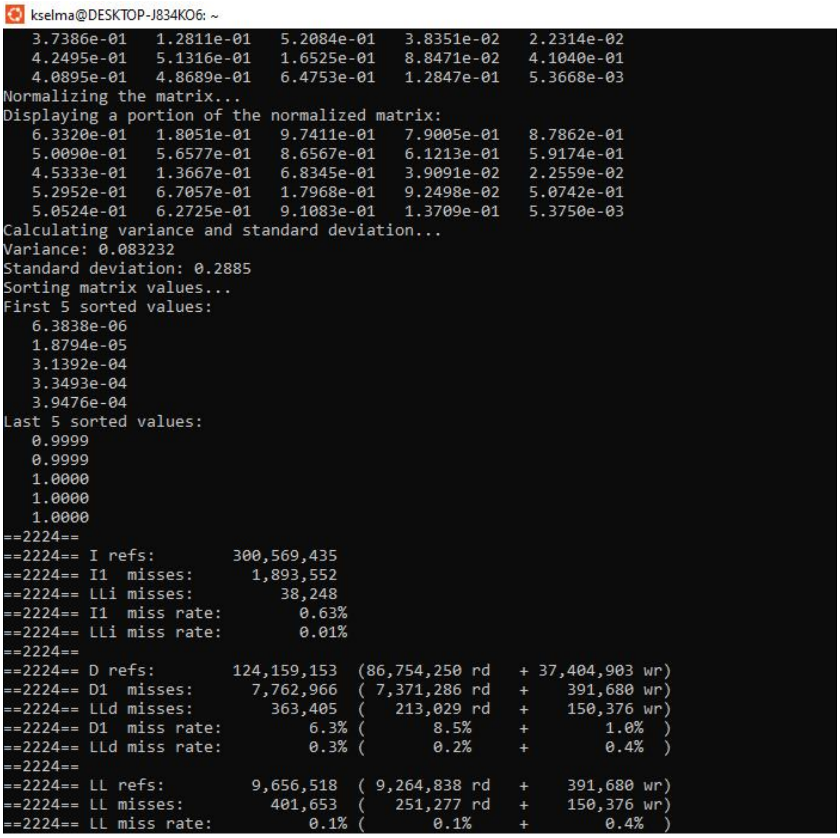
```
==1970== Cachegrind, a high-precision tracing profiler
==1970== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==1970== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==1970== Command: abiword
==1970==
--1970-- warning: L3 cache found, using its data for the LL simulation.

** (abiword:1970): WARNING **: 18:08:14.009: Running under buggy valgrind, see http://bugs.kde.org/show_bug.cgi?id=164298
==1970== brk segment overflow in thread #1: can't grow to 0x484d000
==1970== (see section Limitations in user manual)
==1970== NOTE: further instances of this message will not be shown

(abiword:1970): Gtk-CRITICAL **: 18:08:44.782: gtk_render_background: assertion 'GTK_IS_STYLE_CONTEXT (context)' failed

(abiword:1970): Gtk-CRITICAL **: 18:08:44.787: gtk_render_frame: assertion 'GTK_IS_STYLE_CONTEXT (context)' failed
==1970==
==1970== I refs:        2,737,543,645
==1970== I1  misses:       27,164,364
==1970== LLi misses:          309,094
==1970== I1  miss rate:          0.99%
==1970== LLi miss rate:          0.01%
==1970==
==1970== D refs:        1,061,097,731  (723,894,160 rd   + 337,203,571 wr)
==1970== D1  misses:       34,894,982  ( 30,762,117 rd   +   4,132,865 wr)
==1970== LLd misses:        1,181,096  (    793,390 rd   +     387,706 wr)
==1970== D1  miss rate:           3.3% (        4.2%     +         1.2%  )
==1970== LLd miss rate:           0.1% (        0.1%     +         0.1%  )
==1970==
==1970== LL refs:          62,059,346  ( 57,926,481 rd   +   4,132,865 wr)
==1970== LL misses:         1,490,190  (  1,102,484 rd   +     387,706 wr)
==1970== LL miss rate:            0.0% (        0.0%     +         0.1%  )
```

Result for Data Processing program



```
kselma@DESKTOP-J834KO6: ~
   3.7386e-01   1.2811e-01   5.2084e-01   3.8351e-02   2.2314e-02
   4.2495e-01   5.1316e-01   1.6525e-01   8.8471e-02   4.1040e-01
   4.0895e-01   4.8689e-01   6.4753e-01   1.2847e-01   5.3668e-03
Normalizing the matrix...
Displaying a portion of the normalized matrix:
   6.3320e-01   1.8051e-01   9.7411e-01   7.9005e-01   8.7862e-01
   5.0090e-01   5.6577e-01   8.6567e-01   6.1213e-01   5.9174e-01
   4.5333e-01   1.3667e-01   6.8345e-01   3.9091e-02   2.2559e-02
   5.2952e-01   6.7057e-01   1.7968e-01   9.2498e-02   5.0742e-01
   5.0524e-01   6.2725e-01   9.1083e-01   1.3709e-01   5.3750e-03
Calculating variance and standard deviation...
Variance: 0.083232
Standard deviation: 0.2885
Sorting matrix values...
First 5 sorted values:
   6.3838e-06
   1.8794e-05
   3.1392e-04
   3.3493e-04
   3.9476e-04
Last 5 sorted values:
   0.9999
   0.9999
   1.0000
   1.0000
   1.0000
==2224==
==2224== I refs:        300,569,435
==2224== I1  misses:      1,893,552
==2224== LLi misses:         38,248
==2224== I1  miss rate:        0.63%
==2224== LLi miss rate:        0.01%
==2224==
==2224== D refs:        124,159,153  (86,754,250 rd   + 37,404,903 wr)
==2224== D1  misses:      7,762,966  ( 7,371,286 rd   +    391,680 wr)
==2224== LLd misses:        363,405  (   213,029 rd   +    150,376 wr)
==2224== D1  miss rate:         6.3% (       8.5%     +        1.0%  )
==2224== LLd miss rate:         0.3% (       0.2%     +        0.4%  )
==2224==
==2224== LL refs:         9,656,518  ( 9,264,838 rd   +    391,680 wr)
==2224== LL misses:         401,653  (   251,277 rd   +    150,376 wr)
==2224== LL miss rate:          0.1% (       0.1%     +        0.4%  )
```

Data Processing Configuration:

Result for Game program

```
                   _________
                  (         )
         O       ( Simon says )
        /|\       (_________)
        / \
                     Your turn (Press Enter to exit): a









          Game Over!
          Final Score: 0
          Press any key to exit...==8020==
==8020== I refs:        180,350
==8020== I1  misses:      1,398
==8020== LLi misses:      1,374
==8020== I1  miss rate:    0.78%
==8020== LLi miss rate:    0.76%
==8020==
==8020== D refs:         61,930  (43,265 rd   + 18,665 wr)
==8020== D1  misses:      1,698  ( 1,305 rd   +    393 wr)
==8020== LLd misses:      1,432  ( 1,069 rd   +    363 wr)
==8020== D1  miss rate:     2.7% (   3.0%     +    2.1%  )
==8020== LLd miss rate:     2.3% (   2.5%     +    1.9%  )
==8020==
==8020== LL refs:         3,096  ( 2,703 rd   +    393 wr)
==8020== LL misses:       2,806  ( 2,443 rd   +    363 wr)
```

Result for Word Processing program

```
brki@DESKTOP-5REVLB6:~$ valgrind --tool=cachegrind
--cache-sim=yes abiword
==8182== Cachegrind, a high-precision tracing profiler
==8182== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==8182== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==8182== Command: abiword
==8182==
--8182-- warning: L3 cache found, using its data for the LL simulation.

** (abiword:8182): WARNING **: 17:50:18.744: Running under buggy valgrind, see http://bugs.kde.org/show_bug.cgi?id=164298
==8182== brk segment overflow in thread #1: can't grow to 0x4853000
==8182== (see section Limitations in user manual)
==8182== NOTE: further instances of this message will not be shown

(abiword:8182): Gtk-CRITICAL **: 17:50:29.408: gtk_render_background: assertion 'GTK_IS_STYLE_CONTEXT (context)' failed

==8182== 8182): Gtk-CRITICAL **: 17:50:29.410: gtk_render_frame: assertion 'GTK_IS_STYLE_CONTEXT (context)' failed
==8182== I refs:        4,910,450,775
==8182== I1  misses:       64,093,942
==8182== LLi misses:          701,252
==8182== I1  miss rate:          1.31%
==8182== LLi miss rate:          0.01%
==8182==
==8182== D refs:        2,008,283,403  (1,313,698,943 rd   + 694,584,460 wr)
==8182== D1  misses:       50,460,555  (   44,855,617 rd   +   5,604,938 wr)
==8182== LLd misses:        1,217,856  (      714,362 rd   +     503,494 wr)
==8182== D1  miss rate:           2.5% (          3.4%     +         0.8%  )
==8182== LLd miss rate:           0.1% (          0.1%     +         0.1%  )
==8182==
==8182== LL refs:         114,554,497  (  108,949,559 rd   +   5,604,938 wr)
==8182== LL misses:         1,919,108  (    1,415,614 rd   +     503,494 wr)
==8182== LL miss rate:            0.0% (          0.0%     +         0.1%  )
brki@DESKTOP-5REVLB6:~$ |
```

Result for Data Processing program

```
Calculating basic statistics...
Mean value: 0.49923
Max value: 0.99974
Min value: 7.2994e-05
Calculating row and column sums...
First 5 row sums:
   54.766
   46.833
   49.576
   50.609
   47.114
First 5 column sums:
   48.046   44.568   50.729   54.654   49.658
Applying element-wise transformation...
Displaying a portion of the transformed matrix:
   0.223835   0.679239   0.055718   0.625973   0.535806
   0.453193   0.325878   0.603671   0.485383   0.192664
   0.531135   0.493515   0.194088   0.687580   0.657268
   0.356997   0.263472   0.085020   0.295122   0.673782
   0.461142   0.426714   0.588200   0.405054   0.429013
Normalizing the matrix...
Displaying a portion of the normalized matrix:
   0.250876   0.972632   0.057246   0.870286   0.708992
   0.573448   0.385304   0.829027   0.624936   0.212474
   0.701025   0.638207   0.214203   0.989158   0.929754
   0.429104   0.301470   0.088696   0.343333   0.961895
   0.586010   0.532321   0.800942   0.499479   0.535849
Calculating variance and standard deviation...
Variance: 0.083073
Standard deviation: 0.28822
Sorting matrix values...
First 5 sorted values:
   7.2994e-05
   1.5941e-04
   1.7015e-04
   1.8002e-04
   2.1275e-04
Last 5 sorted values:
   0.9993
   0.9993
   0.9995
   0.9997
   0.9997
octave:35> exit
==8326==
==8326== I refs:        322,079,962
==8326== I1  misses:      1,916,834
==8326== LLi misses:         51,913
==8326== I1  miss rate:        0.60%
==8326== LLi miss rate:        0.02%
==8326==
==8326== D refs:        135,099,794  (91,982,104 rd   + 43,117,690 wr)
==8326== D1  misses:      7,894,563  ( 7,462,938 rd   +    431,625 wr)
==8326== LLd misses:        328,308  (   157,597 rd   +    170,711 wr)
==8326== D1  miss rate:         5.8% (       8.1%     +        1.0%  )
==8326== LLd miss rate:         0.2% (       0.2%     +        0.4%  )
==8326==
==8326== LL refs:         9,811,397  ( 9,379,772 rd   +    431,625 wr)
==8326== LL misses:         380,221  (   209,510 rd   +    170,711 wr)
==8326== LL miss rate:          0.1% (       0.1%     +        0.4%  )
brki@DESKTOP-5REVLB6:~$ |
```

5 Analysis

5.1 Gaming Configuration

For Memcheck:

- **Gaming program:** The gaming configuration showed good memory management with no memory leaks or errors detected by Memcheck. All heap blocks were freed successfully, and the overall memory efficiency was high, ensuring optimal performance for the lightweight gaming application.
- **Word Processing Program:** The word processing program showed significant memory management issues, including invalid reads and writes, along with memory leaks. The heap summary showed a considerable number of bytes still in use at exit, showing potential areas for improvement in memory allocation and deallocation processes. The errors mainly originate from library functions, telling us that dependencies may need debugging or updates.
- **Data Processing Program:** The data processing program showed us efficient memory utilization, as evidenced by Memcheck's report of no critical errors. A minor amount of memory remained reachable at exit, but this does not indicate leaks. Overall, the program's performance and stability were strong, making it highly suitable for intensive computational tasks and data transformations.

For Cachegrind:

- **Gaming Program:** The gaming configuration showed high CPU efficiency with minimal instruction and data cache misses. The miss rates were under 1%, indicating effective use of CPU caches.
- **Word Processing program:** The word processing program revealed substantial CPU utilization with high instruction references and relatively low miss rates (less than 0.1%). However, the presence of GTK-related errors suggests potential inefficiencies in GUI rendering, which could affect overall responsiveness.
- **Data Processing Configuration:** The data processing configuration showed us the CPU and memory usage, with significant instruction and data references due to the computational intensity of matrix operations. Despite this, the cache miss rates were impressively low (below 1%), demonstrating efficient cache utilization. This configuration is well-suited for handling data-intensive workloads.

5.2 Word Processing Configuration

For Memcheck:

- **Gaming Configuration:** The gaming program demonstrated optimal memory usage, with no leaks detected by Memcheck. All heap blocks were freed successfully, indicating efficient memory management. This configuration is well-suited for lightweight tasks, ensuring stability and high performance.

- **Word Processing Configuration:** The word processing configuration revealed significant memory management issues, including uninitialized memory usage, invalid reads/writes, and multiple leaks. While functional, these issues indicate a need for better memory handling, especially when dealing with external libraries. Improvements could enhance stability and efficiency for this medium-demand task.

- **Data Processing Configuration:** The data processing configuration exhibited efficient memory usage, with no critical leaks or errors reported. However, a small amount of memory was still reachable at exit, which is not indicative of significant issues. This setup is robust and performs well under computationally intensive operations, making it suitable for data-heavy tasks.

For Cachegrind

- **Game Program:** The gaming program showed strong performance, with minimal cache miss rates. Instruction cache misses were under 1%, and data cache misses were slightly higher at 2.7%, indicating efficient cache utilization for this lightweight task. CPU utilization was stable, and the system demonstrated optimal efficiency for gaming.

- **Word Processing Program:** The word processing program exhibited high CPU utilization with a substantial number of instruction and data references. Instruction cache miss rates were below 1%, but GUI-related GTK errors indicated inefficiencies in rendering processes, which impacted overall responsiveness. Cache performance remained adequate for the task, though improvements in handling external dependencies could further optimize performance.

- **Data Processing Program:** The data processing program handled computationally intensive operations effectively, with low instruction cache miss rates (0.63%). However, the higher data cache miss rate (6.3%) reflected the demands of matrix computations. Despite this, CPU performance and memory usage were robust, making the configuration suitable for data-heavy workloads. Minor optimizations in memory access patterns could enhance efficiency further.

5.3 Data Processing Configuration

For Memcheck

- **Game program:** The game program exhibited perfect memory efficiency, with no errors or memory leaks detected. All heap blocks were freed successfully, showcasing optimal memory management for this lightweight and straightforward task.

- **Word Processing program:** The word processing program showed significant memory usage, with multiple memory-related issues such as uninitialized value usage, invalid reads/writes, and memory leaks. Despite the program's functionality, these errors indicate the need for optimization, particularly in managing external library dependencies.

- **Data Processing program:** The data processing program demonstrated excellent memory efficiency, with no major errors or leaks reported. Minor reachable memory at exit was observed, but this does not impact performance. The setup is highly robust and well-suited for handling intensive computational tasks with high memory demands.

For Cachegrind

- **Game program:** The gaming program demonstrated efficient CPU utilization with minimal cache miss rates. Instruction cache misses were under 1%, and data cache misses were slightly higher at 2.7%, indicating effective but not perfect cache performance.

- **Word Processing program:** The word processing program showed high instruction and data references due to GUI rendering tasks, with instruction cache miss rates below 1%. However, GTK-related errors indicate inefficiencies within the rendering process.

- **Data Processing program:** The data processing program exhibited the highest workload, with significant instruction and data references. While instruction cache miss rates were low (0.63%), data cache miss rates were higher at 6.3%, reflecting the computational intensity of the operations.

6 Recommendations

Based on the profiling results, these are the recommendations that we would suggest for each computer:

The **gaming configuration**, equipped with an Intel Core i5-10300H CPU (2.5-4.5 GHz), 8GB DDR4 RAM, and an 8MB L3 cache, performed exceptionally well for lightweight gaming tasks. Its high clock speed and efficient cache utilization ensured smooth gameplay, while the dedicated NVIDIA GeForce GTX 1650 provided robust graphical performance. However, the 8GB RAM may become a limiting factor for more demanding games. Upgrading to 16GB or 32GB RAM and further optimizing cache utilization could enhance the system's ability to handle resource-intensive gaming applications.

The **word processing configuration**, featuring an Intel Core i3-3217U CPU (1.8 GHz, dual-core), 6GB DDR4 RAM, and a 3MB L3 cache, faced performance challenges due to its limited CPU power and small cache size. While the cache miss rates were low, memory management issues, including invalid reads/writes and memory leaks, significantly impacted performance. GTK-related rendering inefficiencies further slowed responsiveness. Upgrading to a faster quad-core CPU with a larger cache and increasing the RAM to 8GB or more would address these bottlenecks and improve performance for multitasking and GUI-intensive applications.

The **data processing configuration**, powered by an AMD Ryzen 7 6800H CPU (3.2 GHz), 16GB DDR5-4800 MHz RAM, and a 16MB L3 cache, excelled under heavy workloads. It handled computationally intensive tasks efficiently, with low instruction cache miss rates below 1%, showcasing excellent memory bandwidth and cache performance. However, a higher data cache miss rate (6.3%) suggests room for improvement. Optimizing memory access patterns and algorithms could reduce bottlenecks, further enhancing processing speed and efficiency for data-heavy applications.

Overall, the configurations were well-optimized for lightweight gaming and computationally intensive tasks, but addressing inefficiencies in memory management, increasing RAM speed, and improving cache utilization would enable it to handle a broader range of applications with greater efficiency and stability.

7 Conclusion

Through our analysis, we observed varying levels of efficiency across the three configurations. As second-year students approaching this theme for the first time, conducting this research significantly deepened our understanding of system performance and optimization. The gaming configuration demonstrated excellent memory management and CPU utilization, making it highly effective for lightweight gaming tasks. However, we realized that increasing the memory to 16GB or 32GB and optimizing cache utilization could further enhance its ability to handle more demanding applications.

The word processing configuration, while functional, presented several challenges. We identified inefficiencies in memory management, such as invalid reads, writes, and memory leaks. Additionally, GUI-related inefficiencies in rendering highlighted the need for better handling of library dependencies and improved memory optimization. We learned that upgrading to a faster CPU and increasing memory could resolve these issues and significantly improve its performance.

The data processing configuration impressed us with its ability to handle computationally intensive tasks effectively. It showcased excellent memory efficiency and low instruction cache miss rates. However, the higher data cache miss rates taught us about the importance of memory access patterns and algorithms. We concluded that upgrading to faster RAM and refining memory operations would enhance its performance for large-scale data processing tasks.

Overall, this project helped us gain valuable insights into the importance of tailoring system configurations to specific use cases. By addressing identified bottlenecks, configurations can achieve greater performance, stability, and efficiency. This hands-on research not only improved our technical understanding but also demonstrated the critical impact of optimization in real-world applications.