

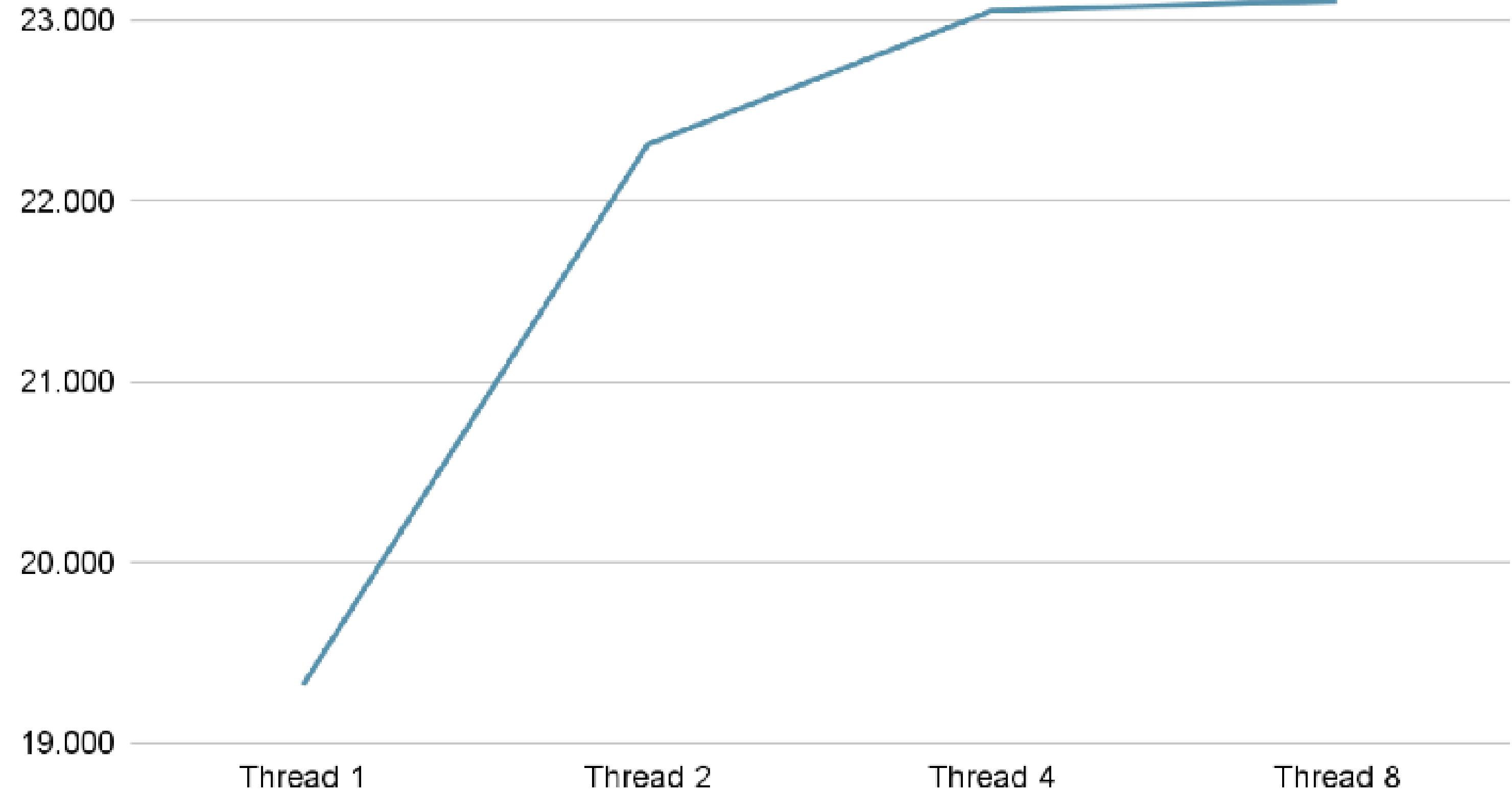
PEDRO MONIZ CANTO
10426546

LÍVIA NEGRUCCI CANTOWITZ
10389419

Nº de Threads(N)	Tempo de Execução (s)	Speedup (S)	Eficiência (E)
1	0.1542 segundos	1,9319	1,9319
2	0.1335 segundos	2,2315	1,11575
4	0.1292 segundos	2,3057	0,576425
8	0.1289 segundos	2,3111	0,2888875

Speedup = Tempo Sequencial / Tempo Paralelo

Eficiência = Speedup / N° de Threads



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <time.h>
5
6 // guarda estatísticas do log
7 typedef struct {
8     long long errors404;
9     long long total_bytes;
10 } Stats;
11
12 // processa uma linha do log e atualiza o stats
13 void parse_log_line(char *line, Stats *stats) {
14     char *quote_ptr = strstr(line, "\" ");
15     if (quote_ptr) {
16         int status_code;
17         long long bytes_sent;
18         if (sscanf(quote_ptr + 2, "%d %lld", &status_code, &bytes_sent) == 2) {
19             if (status_code == 404) {
20                 stats->errors404++;
21             } else if (status_code == 200) {
22                 stats->total_bytes += bytes_sent;
23             }
24         }
25     }
26 }
27
28 int main(int argc, char *argv[]) {
29     // verifica se passou o arquivo como argumento
30     if (argc != 2) {
31         fprintf(stderr, "Uso: %s <caminho_do_arquivo_de_log>\n", argv[0]);
32         return 1;
33     }
34
35     const char *filename = argv[1]; // abre o arquivo
36     FILE *fp = fopen(filename, "r");
37     if (fp == NULL) {
38         perror("Erro ao abrir o arquivo");
39         return 1;
40     }
41
42     struct timespec start, end; // mede o tempo
43     clock_gettime(CLOCK_MONOTONIC, &start); // hora inicial
44
45     Stats stats = {0, 0};
46     char *line = NULL;
47     size_t len = 0;
48
49     while (getline(&line, &len, fp) != -1) {
50         parse_log_line(line, &stats);
51     }
52 }
```

```
clock_gettime(CLOCK_MONOTONIC, &end); // hora final
double time_spent = (end.tv_sec - start.tv_sec) +
                    (end.tv_nsec - start.tv_nsec) / 1e9;

printf("--- Versão Sequencial ---\n");
printf("Total de erros 404: %lld\n", stats.errors404);
printf("Total de bytes transferidos (código 200): %lld\n", stats.total_bytes);
printf("Tempo de execução: %.4f segundos\n", time_spent); // tempo de execução

fclose(fp);
if (line) {
    free(line);
}

return 0;
```

```
@pmcanto → /workspaces/CompParalela-AnalisadorDeLogs (main) $ ./log_analyzer_seq access_log_large.txt
--- Versão Sequencial ---
Total de erros 404: 99667
Total de bytes transferidos (código 200): 21462058072
Tempo de execução: 4.6543 segundos
```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <pthread.h>
5  #include <time.h>
6
7  // struct pra guardar estatisticas
8  typedef struct {
9      long long errors404;
10     long long total_bytes;
11 } Stats;
12
13 // argumentos passados para cada thread
14 typedef struct {
15     int start_line;
16     int end_line;
17     char **lines;
18 } ThreadArgs;
19
20 Stats global_stats = {0, 0}; // stats globais
21 pthread_mutex_t stats_mutex; // mutex para proteger stats globais
22
23 // funcao que cada thread vai executar
24 void* process_log_chunk(void* arg) {
25     ThreadArgs* args = (ThreadArgs*)arg;
26     Stats local_stats = {0, 0};
27
28     for (int i = args->start_line; i < args->end_line; i++) { // percorre linhas atribuidas
29         char *line = args->lines[i];
30         char *quote_ptr = strstr(line, "\" ");
31         if (quote_ptr) {
32             int status_code;
33             long long bytes_sent;
34             if (sscanf(quote_ptr + 2, "%d %lld", &status_code, &bytes_sent) == 2) {
35                 if (status_code == 404) {
36                     local_stats.errors404++;
37                 } else if (status_code == 200) {
38                     local_stats.total_bytes += bytes_sent;
39                 }
40             }
41         }
42     }
43     // atualiza stats globais com mutex
44     pthread_mutex_lock(&stats_mutex);
45     global_stats.errors404 += local_stats.errors404;
46     global_stats.total_bytes += local_stats.total_bytes;
47     pthread_mutex_unlock(&stats_mutex);
48
49     return NULL;
50 }

```

```

int main(int argc, char *argv[]) {
    // checa argumentos
    if (argc != 3) {
        fprintf(stderr, "Uso: %s <caminho_do_arquivo_de_log> <numero_de_threads>\n", argv[0]);
        return 1;
    }

    const char *filename = argv[1];
    int num_threads = atoi(argv[2]);

    // le todas as linhas do arquivo para a memoria
    FILE *fp = fopen(filename, "r");
    if (fp == NULL) {
        perror("Erro ao abrir o arquivo");
        return 1;
    }

    char **lines = NULL;
    char *line = NULL;
    size_t len = 0;
    long num_lines = 0;
    while (getline(&line, &len, fp) != -1) {
        lines = realloc(lines, sizeof(char*) * (num_lines + 1));
        lines[num_lines] = strdup(line); // copia linha para a memoria
        num_lines++;
    }
    fclose(fp);
    if (line) free(line);

    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC, &start); // inicia contagem de tempo

    pthread_t threads[num_threads];
    ThreadArgs thread_args[num_threads];
    pthread_mutex_init(&stats_mutex, NULL);

    // divide linhas entre threads
    long lines_per_thread = num_lines / num_threads;
    long remaining_lines = num_lines % num_threads;
    long current_line = 0;

    for (int i = 0; i < num_threads; i++) {
        thread_args[i].lines = lines;
        thread_args[i].start_line = current_line;
        long chunk_size = lines_per_thread + (i < remaining_lines ? 1 : 0);
        thread_args[i].end_line = current_line + chunk_size;
        current_line += chunk_size;

        pthread_create(&threads[i], NULL, process_log_chunk, &thread_args[i]);
    }

    for (int i = 0; i < num_threads; i++) {
        pthread_join(threads[i], NULL);
    }
}

```

```
clock_gettime(CLOCK_MONOTONIC, &end); // termina contagem de tempo
double time_spent = (end.tv_sec - start.tv_sec) +
                    (end.tv_nsec - start.tv_nsec) / 1e9;

printf("--- Versão Paralela (%d threads) ---\n", num_threads);
printf("Total de erros 484: %lld\n", global_stats.errors484);
printf("Total de bytes transferidos (código 288): %lld\n", global_stats.total_bytes);
printf("Tempo de execução: %.4f segundos\n", time_spent);

pthread_mutex_destroy(&stats_mutex); // limpa mutex
for (long i = 0; i < num_lines; i++) {
    free(lines[i]);
}
free(lines);

return 0;
```

```
@pmcanto → /workspaces/CompParalela-AnalisadorDeLogs (main) $ # Teste com 1 thread  
./log_analyzer_par access_log_large.txt 1  
  
# Teste com 2 threads  
./log_analyzer_par access_log_large.txt 2  
  
# Teste com 4 threads  
./log_analyzer_par access_log_large.txt 4  
  
# Teste com 8 threads  
./log_analyzer_par access_log_large.txt 8  
--- Versão Paralela (1 threads) ---  
Total de erros 404: 99667  
Total de bytes transferidos (código 200): 21462058072  
Tempo de execução: 0.2984 segundos  
--- Versão Paralela (2 threads) ---  
Total de erros 404: 99667  
Total de bytes transferidos (código 200): 21462058072  
Tempo de execução: 0.1327 segundos  
--- Versão Paralela (4 threads) ---  
Total de erros 404: 99667  
Total de bytes transferidos (código 200): 21462058072  
Tempo de execução: 0.1329 segundos  
--- Versão Paralela (8 threads) ---  
Total de erros 404: 99667  
Total de bytes transferidos (código 200): 21462058072  
Tempo de execução: 0.1303 segundos
```