



UNIVERSIDADE ESTADUAL DO PARANÁ - CAMPUS APUCARANA

Theo Okagawa Rodrigues

Relatório Técnico - Arquitetura de Comp.

APUCARANA – PR
2024

Theo Okagawa Rodrigues

**Relatório Técnico - Arquitetura e Organização de
Computadores**

Trabalho apresentado à disciplina de
Arquitetura e Organização de Computadores
do curso de Bacharelado em Ciência da
Computação.

Professor: Guilherme Henrique de Souza Nakahata

**APUCARANA – PR
2024**

SUMÁRIO

INTRODUÇÃO	
CAPÍTULO 1: OBJETIVO	
CAPÍTULO 2: RECURSOS E MOTIVAÇÕES.....	
2.1 Motivação	
2.2 Recursos utilizados.....	
2.2.1 Biblioteca time.h	
2.3 Linguagem de Programação	
2.4 Estrutura de Dados.....	
CAPÍTULO 3: RESULTADOS	
CAPÍTULO 4: CONCLUSÃO	

INTRODUÇÃO

Este relatório apresenta uma análise do desempenho da CPU por meio de um benchmark implementado na linguagem de programação C. O principal objetivo é avaliar o tempo de resposta do processador para criação de matrizes e contas matemáticas em grande escala.

CAPÍTULO 1

OBJETIVO

O principal objetivo deste código é testar o desempenho do processador da máquina ao fazer contas aritméticas em grande escala, utilizando-se criação de matrizes também como critério de teste de velocidade.

CAPÍTULO 2

MOTIVAÇÃO E RECURSOS UTILIZADOS

2.1 Motivação

Como citado no capítulo de objetivo do projeto, a motivação consta em medir e calcular o desempenho de uma máquina. Para isso, foi implementado um código que ao ser compilado e rodado gera um log no terminal especificando o tempo utilizado para seguir as instruções no código escritas, como seguido na figura 1.

A screenshot of a terminal window with a dark background. The prompt is a green character. The user has entered a command, and the output shows two lines of timing information: 'Tempo em milisegundos: 3.53' and 'Tempo em segundos: 0.003533'. Below this, the prompt is followed by 'main' and some numbers. A green checkmark is visible on the right side of the terminal window.

```
> ./main
Tempo em milisegundos: 3.53
Tempo em segundos: 0.003533
~/Doc/u/Unespar_actv/2/c/benchmark > main 11 73
```

Figura 1

Assim, com essas informações, ter conhecimento de dados pertinentes sobre a performance da máquina e demais questões para futuros testes de desempenho.

2.2 Recursos utilizados:

Para melhor leitura e codificação, foi utilizado matrizes criadas com tamanhos definidos globalmente através do comando `#define`, biblioteca `time.h`, for loops e funções derivativas da biblioteca `time.h`.

2.2.1 Biblioteca *time*

Com esta biblioteca, incluída por meio do comando `#include`, é possível manipular unidades de tempo, como por exemplo: ano, mês, dia, hora, minutos e segundos. É fundamental possuir o controle de tais informações, pois com ela é possível medir tempo de execução e pulsos de clock do processador.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  #define MATRIZ_X 1000
6  #define MATRIZ_Y 1000
7
8  double benchmark(){
9      double soma = 0;
10     double mat[MATRIZ_X][MATRIZ_Y];
11
12     srand(time(NULL));
13
14     // Preencher matriz com números aleatórios e calcular a soma acumulada
15     for (int i = 0; i < MATRIZ_X; i++){
16         for (int j = 0; j < MATRIZ_Y; j++){
17             double num = rand() / (double)RAND_MAX;
18             soma += num;
19             mat[i][j] = soma; // Armazenar soma acumulada na matriz
20         }
21     }
22
23     clock_t start = clock();
24
25     // Calcular soma dos elementos da matriz
26     for (int i = 0; i < MATRIZ_X; i++){
27         for (int j = 0; j < MATRIZ_Y; j++){
28             for (int k = 0; k < 100; k++){
29                 for (int h = 0; h < 10; h++){ // Aumenta o número de iterações
30                     soma += mat[i][j];
31                 }
32             }
33         }
34     }
35
36     clock_t end = clock();
37
38     return ((double)(end - start) / CLOCKS_PER_SEC) * 1000.0;
39 }
40
41 int main()

```

Figura 2

Como destacado na figura 2, mostra-se exemplos no código fonte de como e onde as funções da biblioteca foram utilizadas, seguidas com comentários explicando brevemente a funcionalidade.

2.3 Linguagem de Programação:

Para este código fonte, foi escolhida a linguagem de programação C. Tal decisão foi tomada pela facilidade e proximidade que a linguagem possui com níveis mais baixos de hardware. Tal proximidade facilita a integração de funções que residem em menor nível computacional, trazendo maior facilidade para a visualização e coleta de dados da máquina.

2.4 Estrutura de dados:

Sob uma perspectiva geral do código fonte, é criada uma estrutura linear de comandos em uma função void com nome “benchmark”, sem interação humana e simples de ser lido e entendido.

É criado uma matriz mat[][] com tamanhos definidos por variáveis globais, de tamanho 1000, MATRIZ_X e MATRIZ_Y, respectivamente representando o número de linhas e colunas de tal matriz.

Para o preenchimento de números, que no futuro serão utilizados para contas aritméticas, é inicializado uma seed para sempre manter os mesmos números aleatórios, mantendo linearmente o mesmo stress computacional em

todas as máquinas testadas. Também com a finalidade de criar um stress linear, a matriz anteriormente citada é preenchida com os números aleatoriamente gerados através de uma função for loop.

Observando a execução da função, o próximo passo é outro for loop, utilizado para somar os números da matriz e armazenar em uma variável soma. Além disso, foram adicionados mais dois for loops alinhados (k e h) dentro do loop mais interno. Isso aumentará significativamente a carga de trabalho da CPU, pois cada elemento da matriz será somado múltiplas vezes. A inclusão desses loops adicionais aumenta o stress computacional, o que pode ser útil para avaliar o desempenho da CPU em condições de carga mais pesada.

Essa abordagem permite uma análise mais detalhada do desempenho da CPU em situações onde a carga de trabalho é mais intensa, fornecendo dados valiosos sobre a capacidade da CPU de lidar com cenários computacionais mais exigentes.

CAPÍTULO 3

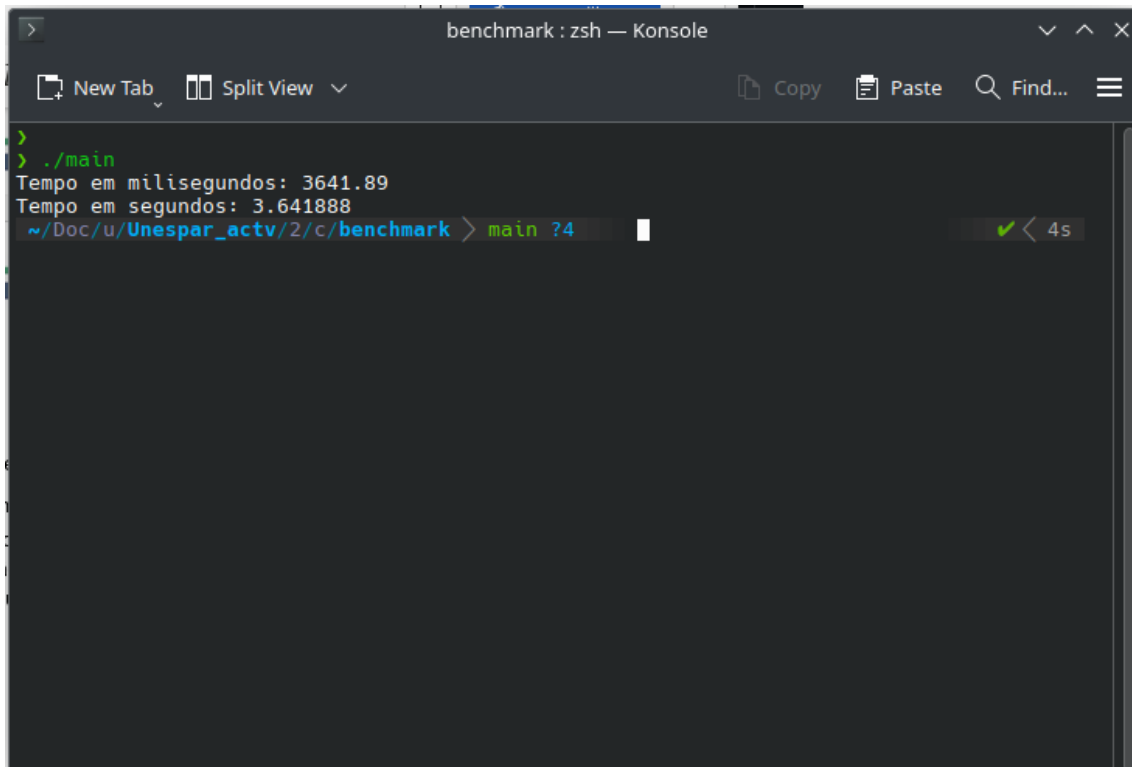
RESULTADOS

Mediante os objetivos e a estrutura apresentada, o resultado mostra-se um código plenamente funcional e que cumpre seu objetivo.

Testes foram feitos em dois sistemas operacionais: Windows 10 e Linux com distribuição Arch, e em ambos o código se apresentou funcional.

Abaixo, nas figuras 3 e 4 podemos ver o funcionamento do benchmark.

Respectivamente, temos o código rodando no terminal do Linux e Windows.



```
benchmark : zsh — Konsole
New Tab Split View
Copy Paste Find...
>
> ./main
Tempo em milisegundos: 3641.89
Tempo em segundos: 3.641888
~/Doc/u/Unespar_actv/2/c/benchmark > main ?4
```

Figura 3



```
Tempo em milisegundos: 2757.02
Tempo em segundos: 2.757019

...Program finished with exit code 0
Press ENTER to exit console.
```


Figura 4

CAPÍTULO 4

CONCLUSÃO

O benchmark implementado proporcionou uma análise do desempenho da CPU ao lidar com tarefas de criação de matrizes e operações aritméticas em grande escala. Através da medição do tempo de resposta do processador, pudemos avaliar a eficiência da máquina em lidar com cargas de trabalho intensas.

Ao utilizar a linguagem de programação C e recursos como a biblioteca `time.h`, foi possível obter dados precisos sobre o tempo de execução das operações, permitindo uma compreensão mais profunda do desempenho do processador em diferentes sistemas operacionais. A escolha da linguagem C foi justificada pela sua proximidade com o hardware e pela facilidade de integração de funções em níveis mais baixos do sistema, garantindo uma análise mais precisa e confiável.

Os resultados obtidos demonstraram que o código desenvolvido foi funcional e eficaz, sendo testado com sucesso em sistemas operacionais como o Windows 10 e o Linux. Através das figuras apresentadas, foi possível visualizar o funcionamento do benchmark em ambos os ambientes, confirmando sua capacidade de execução em diferentes plataformas.

Em suma, este relatório forneceu uma avaliação do desempenho da CPU através do benchmark implementado, destacando sua utilidade na análise de sistemas computacionais e na identificação de possíveis melhorias de performance. Os dados coletados servem como base para futuros testes de desempenho e otimizações, contribuindo para a melhoria contínua da eficiência dos sistemas computacionais.