Sistemas Paralelos e Distribuídos

Engenharia Informática

DTW ALGORITHM REPORT

Ano Letivo 2022/2023

Alunos:
Alexandre Guerreiro, nº 71372
Alexandre Gregório, nº 71373
Leonardo Moreira, nº 71512
Diogo Parrinha, nº 72683
Diogo Catarino, nº 72730

Índice

Resumo

Este relatório tem como objetivo explorar o desenvolvimento de um programa que mede a semelhança entre duas séries temporais. O foco deste projeto está no design de programas paralelos, com o objetivo de melhorar o desempenho e reduzir o tempo de execução. Ao aproveitar técnicas de processamento paralelo, esperamos obter cálculos mais eficientes e escaláveis. O relatório discutirá os detalhes técnicos do design e implementação do programa, incluindo os algoritmos utilizados, os requisitos de hardware e software e as métricas de desempenho medidas. No geral, o relatório visa fornecer informações sobre os benefícios da programação paralela para a análise de séries temporais e o potencial para pesquisas futuras nesta área.

Abstract

This report presents the development of a parallel program for measuring the similarity between two time series, with a focus on improving performance and reducing execution time. The program design incorporates parallel processing techniques to achieve efficient and scalable computations. The report discusses the technical details of the program's algorithms, hardware and software requirements, and performance metrics. The results demonstrate the benefits of parallel programming for time series analysis, highlighting the potential for future research in this area.

1. Introdução

1.1. Objetivo

O objetivo deste relatório é explorar o conceito de Dynamic Time Warping (DTW) e seu uso como objetivo funcional para medir a similaridade entre duas séries temporais. Além disso, o relatório tem como objetivo focar no design de programas paralelos para implementar o DTW de maneira mais eficiente. O DTW é um método comumente usado para medir a similaridade entre duas séries temporais que podem variar em velocidade ou tempo. Ele alinha as duas séries temporais e encontra o caminho ótimo de similaridade entre elas. Este método é amplamente utilizado em campos como processamento de sinais, reconhecimento de fala e bioinformática. O uso de técnicas de programação paralela pode melhorar significativamente a eficiência do cálculo do DTW, especialmente para grandes conjuntos de dados. Ao projetar programas que podem distribuir a carga de trabalho em vários processadores ou núcleos, podemos reduzir o tempo de processamento geral e melhorar o desempenho do algoritmo. Portanto, este relatório cobrirá os conceitos básicos do DTW, seu uso como objetivo funcional para medir a similaridade e o design de programas paralelos para implementá-lo de forma eficiente. Ao final do relatório, os leitores deverão ter uma compreensão sólida do DTW e ser capazes de projetar e implementar programas paralelos para calculá-lo de forma eficiente.

1.2. Motivação

A motivação deste relatório é fornecer uma compreensão abrangente do Dynamic Time Warping (DTW) e o seu uso como objetivo funcional para medir a semelhança entre duas séries temporais. O DTW tornou-se cada vez mais relevante em vários campos devido à crescente disponibilidade de dados de séries temporais. A motivação para explorar técnicas de programação paralela para o cálculo do DTW surge da necessidade de melhorar a sua eficiência para conjuntos de dados grandes. Com a crescente demanda pelo processamento de grandes quantidades de dados, a capacidade de projetar e implementar programas paralelos eficientes tornou-se uma habilidade crítica para pesquisadores e profissionais em diversos domínios. Além disso, este relatório tem como objetivo fornecer um guia prático para projetar programas paralelos para calcular o DTW de forma eficiente. Isso não apenas beneficiará os pesquisadores, mas também os profissionais que precisam aplicar o DTW a problemas do mundo real. Além disso, compreender o DTW e as técnicas de programação paralela pode levar ao desenvolvimento de novos algoritmos e aplicações em diversos domínios, como bioinformática, processamento de sinais, reconhecimento de fala e muitos outros.

1.3. Organization of the report

2. Enquadramento

2.1. Caso a estudar

To be done

2.2. Métodos e técnicas

To be done

2.3. Métricas de avaliação de desempenho

To be done

2.4. Descrição das máquinas

Computador 1:

Nome do dispositivo LAPTOP-4DQABV65

Processador Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz

RAM instalada16,0 GB (15,8 GB utilizável)

ID do Dispositivo910E1C1D-5475-460B-A8AD-785CA2A7D341

ID do Produto00325-81794-25160-AAOEM

Tipo de sistema Sistema operativo de 64 bits, processador baseado em x64

Computador 2:

Nome do dispositivo macOS Big Sur

Versão 11.61

MacBook Air

Processador 1,3 GHz Intel Core i5 de núcleo duplo

Memória 8GB1600 MHz DDR3

Placa gráfica Intel HD Graphics 5000 1536 MB.

Número de série CO2L354CF6T5

3. Estudos de Casos

3.1. Metodologia

To be done

3.2. Implementação Sequencial

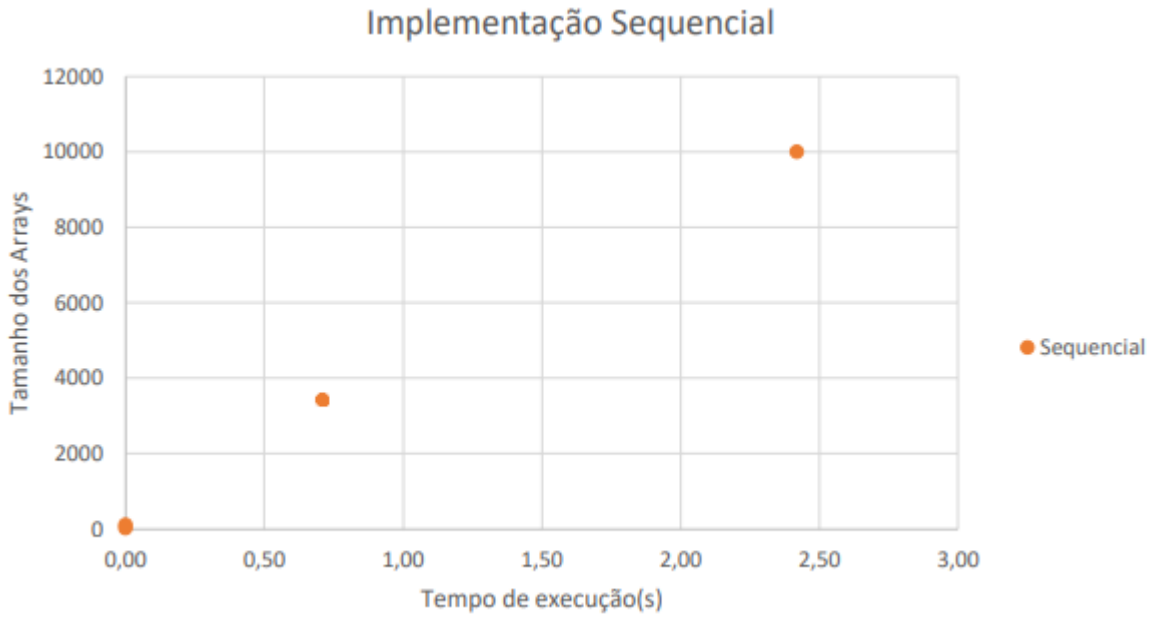
```c
int DTW(int a[], int size_a, int b[], int size_b) {

    int** DTW = (int**)malloc((size_a+1) * sizeof(int*));
    for (int i = 0; i <= size_a; i++) {
        DTW[i] = (int*)malloc((size_b+1) * sizeof(int));
    }

    int i, j, cost;

    for (i = 1; i <= size_a; i++)
        DTW[i][0] = INT_MAX;
    for (i = 1; i <= size_b; i++)
        DTW[0][i] = INT_MAX;
    DTW[0][0] = 0;

    for (i = 1; i <= size_a; i++) {
        for (j = 1; j <= size_b; j++) {
            cost = abs(a[i-1] - b[j-1]);
            DTW[i][j] = cost + fmin(DTW[i-1][j], fmin(DTW[i][j-1], DTW[i-1][j-1]));
        }
    }

    return DTW[size_a][size_b];
}
```

3.4. Implementação usando o OpenMP

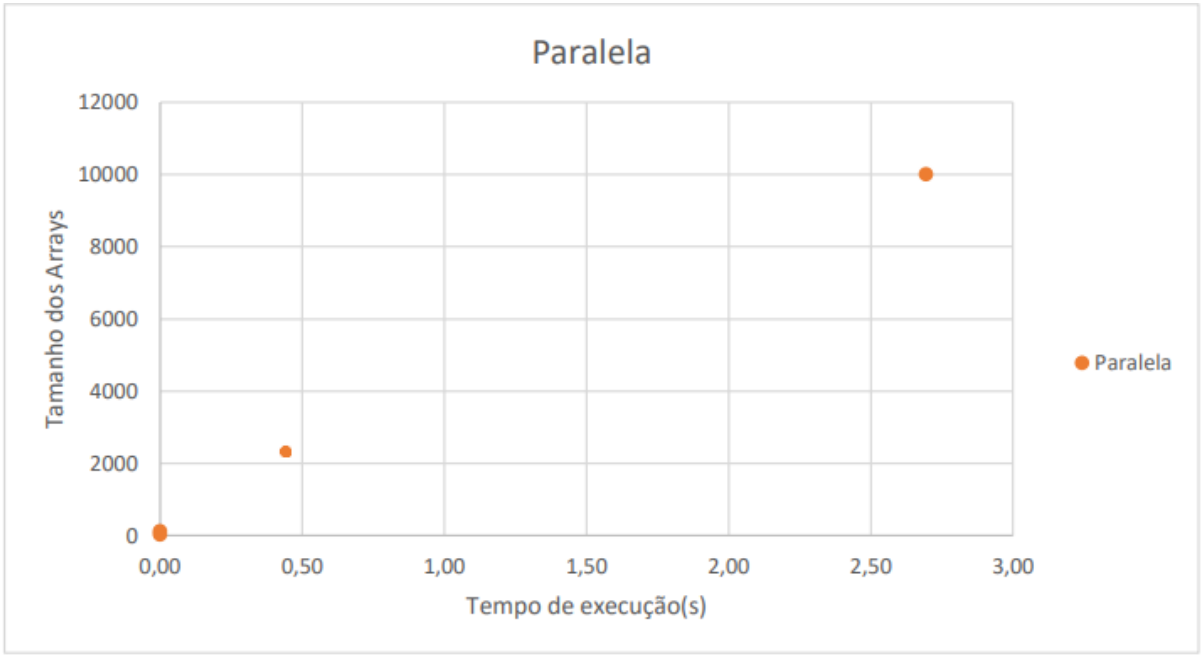
```
int DTWP(int a[], int size_a, int b[], int size_b) {
    int** DTW = (int**)malloc((size_a+1) * sizeof(int*));
    for (int i = 0; i <= size_a; i++) {
        DTW[i] = (int*)malloc((size_b+1) * sizeof(int));
    }

    int i, j, cost;


    for (i = 1; i <= size_a; i++)
        DTW[i][0] = INT_MAX;
    for (i = 1; i <= size_b; i++)
        DTW[0][i] = INT_MAX;
    DTW[0][0] = 0;


    int nthreads, tid;
    #pragma omp parallel shared(DTW) num_threads(THREADS)
    {

        /*#pragma omp master
        {
            nthreads = omp_get_num_threads();
            printf("Running with %d threads\n", nthreads);
        }*/

        #pragma omp for ordered
        for (i = 1; i <= size_a; i++) {
            #pragma omp ordered
            for (j = 1; j <= size_b; j++) {
                cost = abs(a[i-1] - b[j-1]);
                DTW[i][j] = cost + fmin(DTW[i-1][j], fmin(DTW[i][j-1], DTW[i-1][j-1]));
            }
        }
    }

    return DTW[size_a][size_b];
}
```

4. Avaliação de desempenho

4.1 Implementação Sequencial



4.2. Implementação utilizando OpenMP



5. Análise de resultados

5.1. Implementação OpenMP

5.4. Discussão sobre os resultados obtidos

6. Conclusão

Este trabalho teve como finalidade desenvolver a nossa capacidade de trabalho com OpenMP usando ferramentas de paralelização para otimizar o algoritmo Dynamic Time Warping. No entanto, após a realização dos testes concluimos que a implementação paralela não teve grande impacto na performance e acabou por piorar a performance em certas situações, Acreditamos que isto possa ter sido devido a um erro na implementação do algoritmo.