TDE Paralelo

Organização de Computadores - CC53B

Prof. Dr. Erikson Freitas de Morais
Thiago G. Belem - 2459728

Resumo - Este trabalho propõe a aplicação do paralelismo de hardware usando a biblioteca OpenMP na implementação do algoritmo de Saito e Toriwaki para transformações de distâncias euclidianas em imagens binárias. Foi realizada uma comparação entre a implementação sequencial e paralela, visando analisar o desempenho e a eficiência de cada abordagem.

Index terms - Paralelismo, Scripts, TDE, Organização de Computadores

1. Introdução

A transformação de distâncias euclidianas em imagens binárias é um problema fundamental no processamento de imagens e análise de padrões. Essa técnica permite a representação de informações de uma imagem em formato binário, onde os pixels são classificados como pertencentes a um determinado objeto ou como fundo. Diversas abordagens têm sido propostas para realizar essa transformação, buscando otimizar o desempenho e a precisão dos resultados.

Um dos algoritmos amplamente estudados nesse contexto é o algoritmo de Saito e Toriwaki, que apresenta uma solução eficiente e robusta para a transformação de distâncias euclidianas em imagens binárias. Esse algoritmo utiliza a técnica conhecida como distância euclidiana transformada (TDE), que atribui a cada pixel de uma imagem um valor que representa a menor distância euclidiana entre esse pixel e os pixels pertencentes a um objeto de interesse.

No entanto, apesar da eficácia do algoritmo de Saito e Toriwaki, a aplicação desse método em imagens de grande escala e alta resolução pode ser computacionalmente intensiva e demorada. Uma abordagem promissora para superar essas limitações é a utilização do paralelismo de hardware, aproveitando a capacidade de processamento de sistemas com múltiplos núcleos. Nesse sentido, a biblioteca OpenMP e a linguagem de programação C têm se mostrado ferramentas adequadas para a implementação de soluções paralelas eficientes.

Dessa forma, o objetivo deste trabalho é explorar a aplicação do paralelismo de hardware utilizando a biblioteca OpenMP na implementação do algoritmo de Saito e Toriwaki para transformações de distâncias euclidianas em imagens binárias. Pretende-se analisar o desempenho dessa abordagem paralela em termos de tempo de execução, comparando-o com a implementação sequencial tradicional. Além disso, será realizada uma avaliação da precisão dos resultados obtidos, considerando imagens de diferentes tamanhos e complexidades.

Por fim, a compreensão do impacto do paralelismo de hardware na eficiência da transformação de distâncias euclidianas em imagens binárias é essencial para melhorar a velocidade e escalabilidade desse processo. A investigação desse problema contribui para o avanço do campo de processamento de imagens, fornecendo insights valiosos para aplicações em diversas áreas, como reconhecimento de padrões e visão computacional.

2. Metodologia

Inicialmente, implementamos o algoritmo sequencial em linguagem C, seguindo o pseudo algoritmo fornecido pelo professor[1]. Essa implementação foi dividida em duas transformações distintas de distância. A primeira transformação calcula a menor distância ao quadrado de cada pixel em relação ao pixel de figura mais próximo em uma determinada linha. Já a segunda transformação, utiliza os dados gerados pela primeira transformação e faz o seguinte processo: Para cada valor da primeira transformada em uma coluna, é calculado a distância ao quadrado entre esse item e os demais elementos dessa mesma coluna; Então é realizada uma soma entre os respectivos valores (Primeiro item da coluna, somado à distância entre o item da coluna sendo calculado, e assim por diante). O menor valor de soma é atribuído à segunda transformação no seu correspondente lugar. Mais detalhes em [2], [3].

O algoritmo sequencial foi executado em um único núcleo de processamento, sem a utilização de paralelismo. Posteriormente, para a implementação paralela, foi explorada a biblioteca OpenMP, que permite a criação de programas paralelos em C. A estratégia adotada foi paralelizar os laços *for* que calculavam a primeira e segunda transformada. Diretivas de paralelismo fornecidas pela OpenMP foram utilizadas para distribuir as tarefas entre as threads disponíveis nos sistemas com múltiplos núcleos de processamento. Dessa forma, diferentes threads executaram simultaneamente o cálculo das distâncias euclidianas para suas regiões atribuídas, acelerando o processo de transformação de distâncias euclidianas em imagens binárias.

A diferença entre os programas sequenciais e paralelos está na forma como as tarefas são executadas. No programa sequencial, todas as operações são realizadas em sequência, processando um pixel de cada vez. Já no programa paralelo, existem threads que trabalham em paralelo, processando diferentes regiões da imagem ao mesmo tempo. Isso proporciona uma divisão do trabalho mais eficiente, reduzindo o tempo total de processamento.

A diferença de desempenho entre os programas sequenciais e paralelos pode ser explicada pelo paralelismo de hardware oferecido pelos sistemas com múltiplos núcleos de processamento. Enquanto o programa sequencial depende apenas de um único núcleo para executar todas as tarefas, o programa paralelo pode aproveitar o poder de processamento simultâneo de vários núcleos. Isso resulta em uma distribuição mais equilibrada do trabalho e na redução do tempo de execução.

O programa paralelo tende a ser mais rápido do que o sequencial devido à divisão do trabalho entre as threads e ao aproveitamento eficiente dos recursos de hardware disponíveis. No entanto, é importante destacar que a eficiência do paralelismo depende do número de threads disponíveis e da natureza das tarefas a serem executadas.

3. Experimentos

A máquina utilizada para realização dos experimentos possui um processador Intel(R) Celeron(R) N4000 com 2 Núcleos físicos e 2 Núcleos lógicos. As imagens utilizadas para experimentos, assim como o código fonte do algoritmo e o script em python para geração do gráfico se encontram em um repositório no Github do aluno[3].

Para a obtenção de imagens binárias foi utilizado o software ImageMagick no linux, onde basta um comando no terminal e a conversão era feita[4]. A ideia utilizada para garantir medidas confiáveis, foi utilizar sempre a mesma imagem(disponível em [3] na pasta *images*), porém com diferentes

resoluções como entrada. Assim foi possível manter constância na medição e aumentar o tamanho da entrada.

Ademais, por existir uma variação no tempo de execução do algoritmo, uma medida eficiente para garantir uma média de tempo confiável, foi repetir cada entrada 100 vezes e então calcular o tempo médio dessas 100 entradas.

O processo para obtenção dos valores estatísticos foi o seguinte: Antes da execução das transformadas, uma *array* contendo o caminho para os arquivos *.pbm* de imagem foi inicializada. Além disso, no arquivo do algoritmo paralelo, dois arquivos texto foram abertos para escrita, em um arquivo serão escritos os tempos médios para calcular cada entrada e no outro arquivo é colocado os valores das resoluções das imagens. No algoritmo sequencial a abordagem é a mesma, porém não há a necessidade de reescrever as resoluções de imagem, pois são as mesmas.

Agora que o programa foi executado, existem 3 arquivos texto contendo os tempos médios dos algoritmos(paralelo e sequencial) e o valor das resoluções. Com os dados desses arquivos, é possível executar um script em python que fará estimativas do tempo levado para computação dos algoritmos de acordo com o aumento da entrada; Além de calcular o speedup médio; E tudo isso com uma visualização gráfica.

O processo por trás do script é o seguinte: Primeiro são importadas a bibliotecas matplotlib e numpy; Depois é aberto o arquivo que contém as médias de tempo de execução e é colocado dentro de uma lista seus respectivos valores; Após isso é aberto o arquivo de resoluções, que também vai ser armazenado em uma lista. Agora que os valores necessários para fazer estimativas estão no programa, é feito uma regressão linear de grau 2 para estimar o comportamento da curva que relaciona tempo de execução e resolução da imagem; Processo esse que foi muito facilitado pelas bibliotecas importadas.

O processo da regressão linear é o seguinte: No código, uma nova figura é criada. A regressão polinomial é realizada usando ‘np.polyfit()’ para calcular os coeficientes da curva prevista. Os pontos de dados são plotados e a curva prevista é desenhada usando ‘plt.plot()’. Os eixos do gráfico são rotulados e o gráfico resultante é salvo como uma imagem. Por fim, o gráfico é fechado. Tal processo é utilizado tanto para estimar o algoritmo sequencial, tanto para o paralelo.

O cálculo do speedup é um pouco diferente. O speedup é calculado dividindo cada tempo médio do algoritmo sequencial pelo tempo correspondente do programa paralelo. Em seguida é calculado a média dos speedups obtidos, o que fornece a ideia do speedup médio em relação a todas as resoluções. Uma nova figura é criada para a plotagem. Os valores de speedup são representados com círculos, e o speedup médio é uma linha tracejada vermelha. Segue os gráficos obtidos:

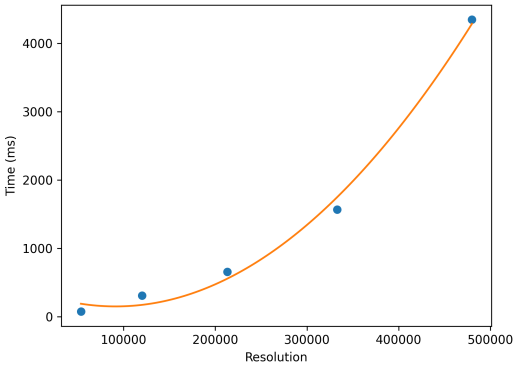

Figura 1. Curva estimada do tempo levado para fazer a TDE de um algoritmo sequencial.

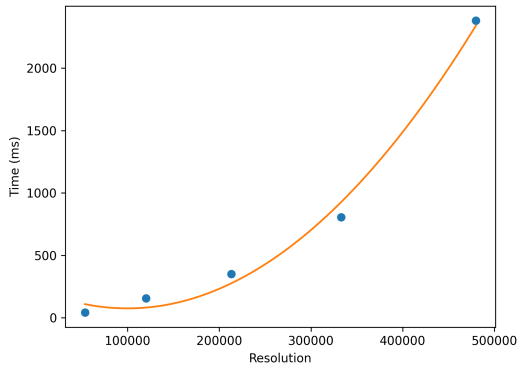

Figura 2. Curva estimada do tempo levado para fazer TDE de um algoritmo paralelo.

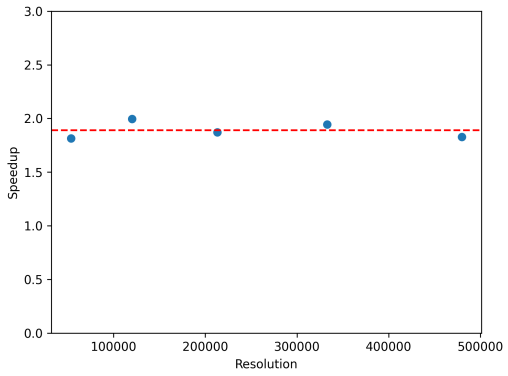

Figura 3. Valores do speedup calculado para cada entrada, além da média em vermelho.

As imagens podem ser vistas de maneira melhor no repositório[3].

4. Conclusão

Agora que se tem todos os dados em mãos, é necessário avaliar se são precisos e representam estatísticas confiáveis. Para melhor compreensão das inferências que serão feitas a seguir, é útil saber algumas informações.

Primeiro, as resoluções das imagens foram, respectivamente: 53400px, 120000px, 213200px, 333000px, 480000px.

Os tempos médios de execução do algoritmo sequencial foram, respectivamente: 76.199ms, 309.516ms 657.443ms, 1568.061ms, 4347.401ms.

Os tempos médios de execução do algoritmo sequencial foram, respectivamente:42.024ms,155.136ms, 351.431ms, 806.629ms, 2379.974ms

É necessário saber também que 2 threads foram fornecidas pelo S.O para a máquina em questão. Além disso, é necessário pontuar que os algoritmos foram compilados com a *flag* -O3.

Tendo tais dados em mente, é possível concluir que os dados representam uma estatística confiável. O algoritmo paralelo utilizou o dobro de threads que o sequencial usou, tornando-o, em teoria 2x mais eficaz. A falta de exatidão no dobro de eficiência se deve ao fato de outros programas poderem estar sendo rodados no fundo e algumas outras interferências. Entretanto, como é possível observar na figura 3, o valor médio do speedup é muito próximo de 2.

Ademais, os gráficos de estimativa do tempo levado para execução, também são confiáveis. Os algoritmos são “iguais”, a curva que define seus tempos não poderia ser diferente. Note que, de acordo com o aumento da entrada, o tempo de computação cresce exponencialmente. O que difere entre os algoritmos, é o tempo levado para computar isso, o paralelo leva basicamente metade do tempo do sequencial. O comportamento exponencial da curva se deve pela complexidade do algoritmo, pois existem *loops* dentro de *loops*, o que acarreta em um incremento significativo a cada aumento no valor da entrada.

5. Ponto bônus

Como a implementação do algoritmo foi compartilhada posteriormente, e o aluno já havia feito sua própria implementação, surge a necessidade de explicar como foi o processo.

Primeiramente, é bom salientar que o código foi escrito em Inglês, pois foi um trabalho extremamente interessante e eu gostaria que houvesse uma maior visibilidade dentro do repositório e foi uma oportunidade de enriquecer meu portfólio.

Na seção de experimentos foi apontado que houve abertura de alguns arquivos, para que a abertura de arquivos *.pbm* fosse bem sucedida, foi necessário estudar sobre sua estrutura, saber seu número mágico e como é formatado; Tal processo usa a função fscanf e garante que o programa só procede se todos os dados estiverem conformes.

Uma matriz de nome *image* é criada, do mesmo tamanho de linhas e colunas do arquivo; Então é preenchida com os dados presentes nele, e o arquivo é fechado.

Uma variável *sumOfTimes* é criada para armazenar a soma de todos os 100 tempos de execução do algoritmo para realizar uma média de tempo de execução confiável. Além de iniciar a marcação de tempo com código fornecido pelo professor.

Agora, para a implementação do algoritmo de Saito e Toriwaki, Uma matriz T1 é criada, com as mesmas dimensões de *image* e inicializada com zeros.

A primeira transformada calcula as distâncias(ao quadrado) mais próximas de todos os itens à figuras na própria linha, para isso existem dois *for's* aninhados, para garantir que toda a matriz será varrida. O for mais externo passa por todas as linhas e o mais interno por todas as colunas.

Entretanto, além de passar por todos os elementos, é necessário calcular a menor distância dos pixels que representam *background* em relação aos que representam figura, para isso é feito um *if*, comparando se a imagem naquele ponto é um 0 (ou seja é background) e então, todos os elementos que estão nessa linha são varridos e mais um *if* é feito, pois se esse ponto nessa linha for figura(é representado com 1), é calculado uma distância, e mais um *if* é feito, para armazenar na matriz T1 a menor distância encontrada nesse ponto nessa linha. Note que não é feito o cálculo para pontos que são figura(igual a 1), pois a matriz T1 já foi inicializada com 0 e a menor distância de um ponto de figura a uma figura é definida como 0.

A TDE calcula a menor distância (ao quadrado) de um ponto à uma figura.

Agora que a primeira transformação foi feita, a matriz TDE é inicializada com zeros e parte-se para o seu cálculo. Para que isso seja possível, varremos a matriz T1, de maneira transposta, ou seja, varremos de coluna em coluna, todas as suas linhas. Para cada elemento definimos que seu valor de soma é 0, e então varremos todos os elementos da sua coluna. Armazenamos em uma variável "p" o valor de T1 naquela linha e naquela coluna. É importante notar que existe um caso de borda, se esse valor armazenado em p for igual a 0, mas na imagem original ele também for, definimos ele com um valor máximo (*width x width)*, pois se 0 for atribuído a ele e os cálculos da TDE forem aplicados, o resultado não sai como esperado. Então a distância é computada e armazenada na soma, se ela representar um valor menor que a soma ou se a soma for 0. Ao final desse processo, ainda é feito um *if* para atribuir o valor da soma à TDE, somente se naquele ponto da imagem, era um pixel de *background*. É finalizado o processo de cálculo do tempo, adicionado a *sumOfTimes* e isso se repete 100 vezes para o cálculo da média. O processo é o mesmo para o algoritmo sequencial e paralelo, o que muda é o uso do *#pragma*.

O script para automatizar os gráficos dos dados está no repositório[3] do aluno. O processo básico por trás dele já foi explicado na seção de experimentos.

6. Referências

[1] MORAIS, Erikson Freitas de. TDE Paralelo. Trabalho de Disciplina.UTFPR, 2023.

[2] SAITO, T., AND TORIWAKI, J.-I. New algorithms for euclidean distance

transformation of an n-dimensional digitized picture with applications. Pattern recognition 27, 11 (1994), 1551–1565.

[3] BELEM, Thiago Guimarães. Repositório de Thiago Guimarães Belem. GitHub. Disponível em: https://github.com/Thiagogob/parallel-programming

[4] IMAGEMAGICK. Command-line tools. Disponível em: https://imagemagick.org/script/command-line-tools.php. Acesso em: 18 jun. 2023.