Programação Paralela
Processamento de imagens

Dionmax Figueiredo Nobre[1]

[1] Instituição de ensino – Universidade do vale de Itajaí (Univali)
Caixa Postal 630 – 88302-901 – Itajaí – SC – Brazil

{dionmax}@edu.unival.br

Abstract. *This article describes the concept of parallel programming for image processing using the OpenCV library to make changes to the image, using processor processing to do multiprocessing instead of graphics card.*

Resumo. *Este artigo descreve o conceito de programação paralela para o processamento de imagens utilizando da biblioteca OpenCV para fazer as alterações na imagem, sendo usado processamento em processador para fazer o multiprocessamento ao invés da placa gráfica.*

1. Introdução

Programação paralela é uma técnica de programação que visa melhorar o desempenho de aplicações fazendo melhor uso do hardware disponível, existem algumas formas de fazer uma aplicação executar de forma paralela. Neste artigo será explicado duas formas diferentes de programação paralela, sendo elas: Programação paralela através do uso do processador, e Programação paralela através do uso da Placa de vídeo dedicada. Nota-se que não será abordado a placa de vídeo integrada ao processador. Para o uso da técnica foi feito o tratamento de imagens, para comparar o ganho ou perda de desempenho usando o processamento paralelo através do processador, e sendo feito um comparativo dos resultados obtidos através dos testes em diferentes imagens.

2. Programação Paralela e o Processamento de imagens.

A programação paralela visa o ganho de desempenho em cima do poder do hardware disponível fazendo com que diferentes partes da aplicação sejam executadas nos diferentes núcleos do processador ou da placa de vídeo, sendo comum atualmente ambos possuir mais de um, isso facilita o uso do paralelismo para o ganho de desempenho em aplicações.

O paralelismo funciona de forma conjunta com as outras partes do programa que está sendo executada, quando melhor for a uniformidade das partes processadas, melhor será o aproveitamento do processamento e assim o melhor ganho de desempenho.

Usar o processamento paralelo na placa de vídeo ou a sigla GPU do inglês “graphical processing unit”, que em tradução livre seria “unidade de processamento gráfico” ,para o tratamento de imagens e vídeo é uma técnica muito comum e usada a um tempo, pois o desempenho obtido é significativo, sendo que, a biblioteca ‘OpenCV’ usada para o tratamento de imagens nesse projeto, já faz o uso dela em suas funções, usando a GPU para obter melhor desempenho, funcionando somente em 64Bits.

Se por motivos inexplicáveis, não for possível usar a GPU para fazer o processamento das imagens, para não afetar o desempenho do computador, compensaria fazer o uso da programação paralela usando o processador ou deixar a biblioteca processar de forma natural usando seus meios para o ganho de desempenho, sendo assim, será tratado nos próximos tópicos.

3. Processando imagens

O experimento teve como objetivo mudar a cor dos pixels de uma imagem, repintando eles uma a um, em uma imagem menor, cortada de uma imagem maior, em outras palavravas, foram pegas imagens de diferentes tamanhos, e elas foram divididas em imagens menores delas mesmo.

Essas imagens menores tiveram seus pixels alterados um a um, para dar o efeito conhecido como “Pixelizado” onde nesse experimento, foi pego o pixel mais distante do centro da imagem, e usados para colorir todo os outros pixels da imagem. Pixels é a menor porção de uma imagem, que é divido comumente em três cores, sendo elas: Vermelho, Verde e Azul formando o conhecido padrão ‘RGB’, a biblioteca OpenCv usa o padrão BGR, porém, são as mesmas cores usadas para formar todas as outras.

Figura 1. Resultados pós processamento

A imagem acima possui uma resolução de 800x800, na qual facilita o processamento, quanto maior a resolução e maior o tamanho da imagem, maior é o tempo de processamento.

Há várias formas de fazer com que a imagem se torne mais ou menos pixelada, pois a técnica que foi usada para pixelar é chamada de ‘Cortes’, como já dito antes, que é cortar a imagem em vários pedaços, e trabalhar em cima deles.

Quanto menor for o corte, maior a pixelização será na imagem e mais rápido será o processamento da imagem, pois será preciso tratar menos imagens, tendo assim, uma nova imagem formada em menos tempo.

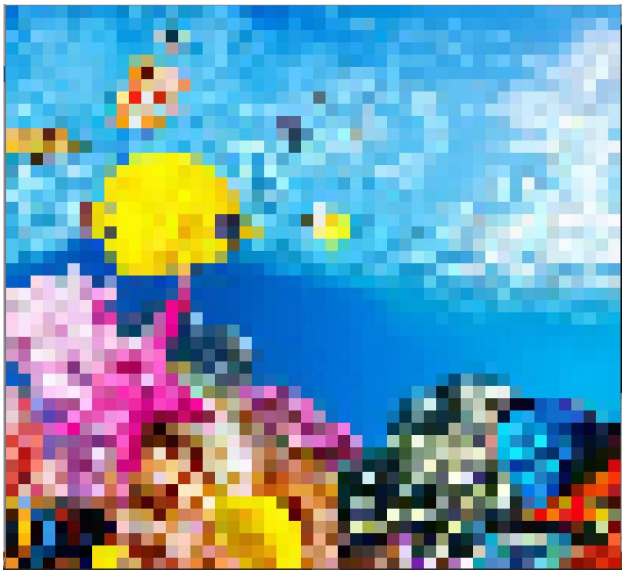

Figura 2. Maior pixelamento da imagem

4. Medindo desempenho

O desempenho foi medido através do tempo de execução começando a partir do momento em que a imagem começa a ser dividida, tratada e novamente unida com a imagem, fazendo uso das próprias ferramentas da linguem para fazer a comparação em tempo de processamento, no qual retorna o tempo em milissegundos (MS).

```
clock_t Ticks[2];

Ticks[0] = clock();

//Execução da Aplicação

Ticks[1] = clock();

double TempoFinal = (Ticks[1] - Ticks[0]) * 1000.0 / CLOCKS_PER_SEC;
```

Figura 3. Código usado para medir o tempo de execução

4.1. Execução serial da aplicação

Diferente do processamento paralelo para executar uma aplicação, a execução em série é a forma como a maioria das aplicações são executadas, na qual o processador executa etapa por etapa, e em certos casos, pode ser mais rápido do que paralelo, que pode ser causa do mal-uso do paralelismo nas aplicações, ou a não necessidade do paralelismo.

Na primeira parte do experimento foi feito o processamento em série da aplicação, na execução mais pesado do código, onde requer maior processamento, são usados dois laços de repetição para fazer as alterações da imagem, sendo quatro laços ao

todo, que estão localizados dentro de outros dois laços de repetição que são usados para realizar o corte da imagem em suas partes menores.

A razão na qual foi usado dois laços de repetição, é dada pelo fato de esta sendo processadas imagens, isso significa que na verdade, estamos lidando com matrizes, onde possuem o eixo X e o eixo Y, e dentro da matriz, esta os valores do RGB.

No primeiro laço de repetição, uma função percorre toda a imagem, obtendo o valor de cada pixel, que armazena em uma variável de um tipo especial da própria biblioteca OpenCv que serve para pegar os três valores de um pixel de canal 3, como já dito antes, o valor do RGB, que variam entre 0 a 255.

No segundo laço de repetição, a cor que foi obtida no laço anterior é aplicada a cada pixel da imagem, lembrando que a imagem é apenas uma pequena parte de uma imagem maior que foi cortada nos dois outros laços de repetição, que são utilizados para cortar a imagem, e que engloba os dois outros.

```
for (int x = 0; x < sub_imagem.rows; x++) {
    for (int y = 0; y < sub_imagem.cols; y++)
    {
        cor = sub_imagem.at<Vec3b>(x, y);
    }
}

for (int x = 0; x < sub_imagem.rows; x++) {
    for (int y = 0; y < sub_imagem.cols; y++)
    {
        sub_imagem.at<Vec3b>(x, y) = cor;
    }
}
```

Figura 4. Código em série usado alterar os pixels da imagem

Nos testes do processamento em série foi usado uma imagem 5000x4000 com o tamanho de 3.100KB~ (3MB), para obter maior uniformidade nos cortes, o teste foi repetido em três vezes, para se obter uma média, em diferentes tamanhos de cortes.

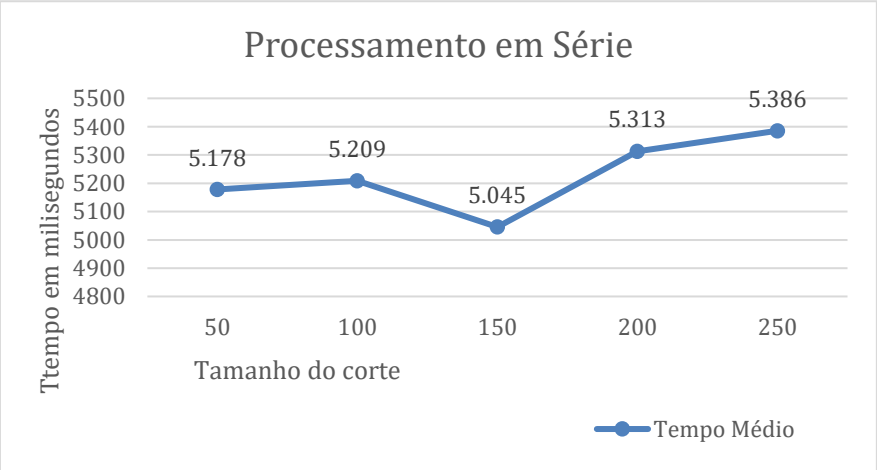

Figura 5. Gráfico mostrando o tempo de execução de acordo com o corte

Como pode ser visto no gráfico acima, o tempo médio de execução foi de cindo mil milissegundos (5000ms), tendo um maior desempenho com o corte de tamanho 150, testes com desempenho muito discrepante foram filtrados, por conta de intervenções de outros processos em segundo plano do computador, que comumente também consome processamento, dando valores falsos-positivos nos testes.

4.2. Execução paralela da aplicação

O processamento paralelo da aplicação, foi feita visando melhor o desempenho dos loops que percorrem os pixels da imagem, e fazem as alterações na cor, pois em teoria, percorrer loops em paralelos se obtém um melhor desempenho.

Para isso, foi paralelizado um bloco contendo os dois loops e mais dois comandos para cada loop individualmente, usado para fazer a melhor divisão de cargas de processamento dentro de um loop.

```
#pragma omp parallel
    {
        for (int x = 0; x < sub_imagem.rows; x++) {
#pragma omp parallel for
            for (int y = 0; y < sub_imagem.cols; y++)
            {
                cor = sub_imagem.at<Vec3b>(x, y);
            }
        }

        for (int x = 0; x < sub_imagem.rows; x++) {
#pragma omp parallel for
            for (int y = 0; y < sub_imagem.cols; y++)
            {
                sub_imagem.at<Vec3b>(x, y) = cor;
            }
        }
    }
```

Figura 6. Código em paralelo usado alterar os pixels da imagem

Nos testes do processamento em paralelo foi usado uma imagem 5000x4000 com o tamanho de 3.100KB~ (3MB), para obter maior uniformidade nos cortes, e o mesmo esquema de coletas de dados foi usado no teste, que foi repetido em três vezes, para se obter uma média, em diferentes tamanhos de cortes.

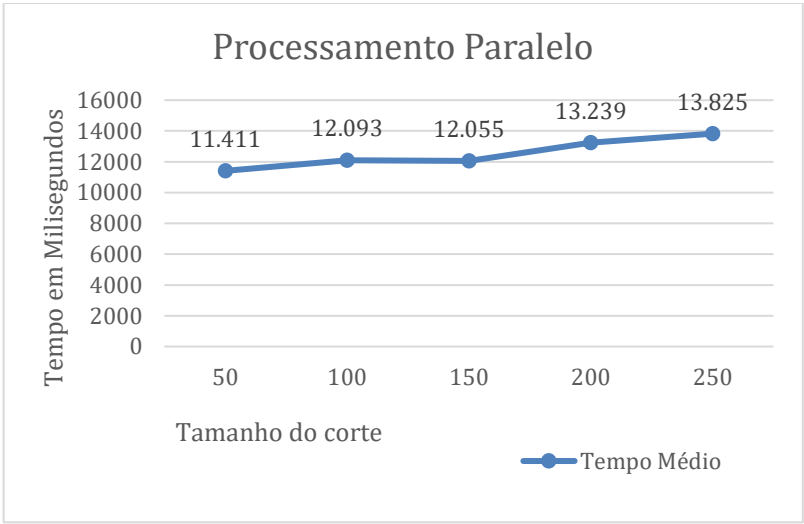

Figura 6. Gráfico mostrando o tempo de execução de acordo com o corte

Como pode ser visto no gráfico acima, o tempo médio variou entre onze mil milissegundos (11000ms) a treze mil milissegundos (13000ms), aumento gradativo conforme o tamanho do corte realizado na imagem. Percebendo que novamente, manteve-se o melhor desempenho com o corte em 150.

5. Comparativo

Um comparativo foi realizado usando os valores obtidos através dos testes de desempenho anteriores, unindo-os e os comparando, para se obter melhor entendimento do quanto de desempenho se obteve ou se perdeu, em questões de tempo de execução, para isso foi usado um gráfico com os valores de cada teste.

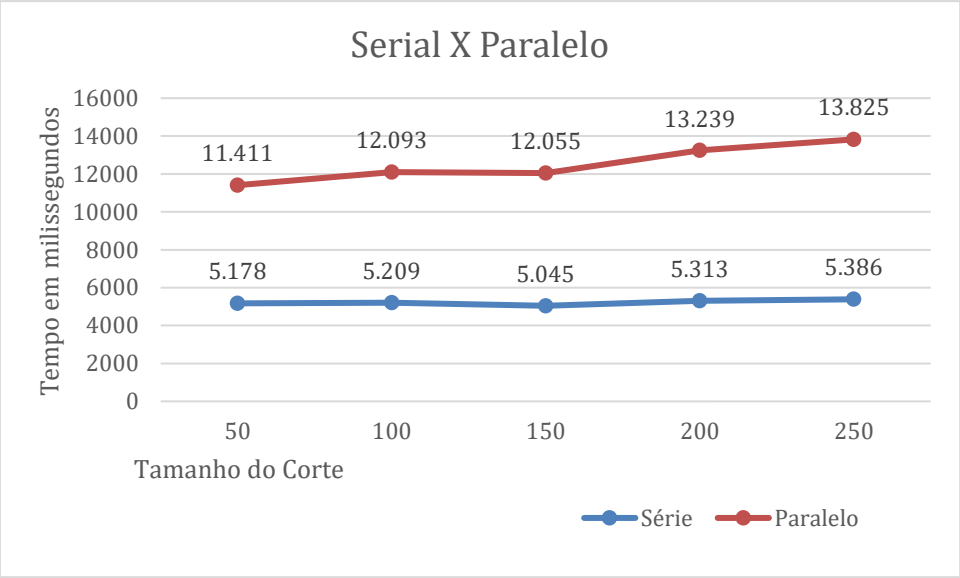

Figura 6. Gráfico mostrando o comparativo entre processamento serial e paralelo

Agora, observando os gráficos acima, será possível perceber que houve uma perda desempenho quando se é usado o paralelismo em processador, isso se deve pelo fato de que imagens são processadas mais rapidamente pelas placas de vídeos gráficas, a biblioteca OpenCv já faz o melhor uso dos recursos gráficos do computador para se obter o melhor desempenho na execução, usando recursos como GPU integrada ou a dedicada para isso.

5.1. Pontos técnicos

A perda de desempenho em relação ao corte de do tamanho 50 foi de 2.20x (120%) em relação a versão não paralelizada do código, com o corte do tamanho 100 foi de 2.32x (132%) pior em relação a versão não paralelizada, com o corte do tamanho 150 foi de 2.38x (138%) pior em relação não paralelizada do código, com o corte de tamanho 200 foi de 2.49x (149%) pior em relação não paralelizada do código, com o tamanho do corte 250 foi de 2.56x (156%) pior em relação a versão não paralelizada.

Outro ponto que vale ressaltar é o hardware que foi usado para realizar os testes foi um laptop com as seguintes configurações: Sistema Operacional Windows 10 Home Single Language, Processador: Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz, 2201 Mhz, 2 Núcleo(s), 4 Processador(es) Lógico(s) na qual significa que foram usados no ao total quatro threads na execução da aplicação, Placa de vídeo dedicada: NVIDIA GeForce 820M e a memória RAM usada foi uma do tipo DIMM_A DDR3 de 8GB.

6. Conclusão

Tendo em vista, todos os dados apresentados, juntamente com os pontos levantados seguindo os resultados dos testes, é possível concluir que a programação paralela em processador não apresenta bons resultados quando se trata de processamento de imagens, sendo recomendando caso você não tenha um grande conhecimento em programação paralela em GPU deixar que a própria linguagem se responsabilize pela execução da aplicação, pois muitas de suas funções nativas já fazem o uso do paralelismo em GPU para se obter melhor desempenho, podendo até forçar o uso usando funções e operações recomendadas pelo criadores do programa.

Por último, o processamento de imagens usado para os testes, foi feito de uma forma simples para que pudesse ser possível obter um melhor conhecimento sobre o assunto de uma forma mais enxuta, existes outras operações que podem ser usadas para medir o desempenho em no paralelismo no processador, como a soma dos valores dos pixels de cada imagem de forma manual, porém esse não era o intuito desse trabalho, que era apenas mostrar se haveria ganho de desempenho fazendo uso do processamento paralelo em processador.

7. Referencias

As referências usadas nesse artigo são de toda e completa autoria de seus criadores, usando apenas para consulta e aplicação dos exemplos fornecidos pelos mesmos que foram eles:

Referencias

OpenCv. Open Source Computer Vision Library. 2018. 1999. Disponível em: <https://opencv.org/>. Acesso em: 04 dez. 2018.

OpenCv. Load and Display an Image. 2018. 1999. Disponível em: <https://docs.opencv.org/2.4/doc/tutorials/introduction/display_image/display_image.html>. Acesso em: 04 dez. 2018.

ATWOOD, Jeff. OpenCV - Mean of Mat object in C++. 2017. 2008. Disponível em: <https://stackoverflow.com/questions/42018164/opencv-mean-of-mat-object-in-c>. Acesso em: 04 dez. 2018.