https://eventos.utfpr.edu.br//sicite/sicite2019

Estudo do paralelismo na construção de mosaicos

Study of parallelism in the construction of mosaics

Othon Alberto da Silva Briganó
othonalberto@gmail.com
Universidade Tecnológica Federal do Paraná Ponta Grossa, Paraná, Brasil

Erikson Freitas de Morais
emorais@utfpr.edu.br
Universidade Tecnológica Federal do Paraná Ponta Grossa, Paraná, Brasil

RESUMO

Pela alta capacidade computacional, o paralelismo é uma ferramenta que se combina perfeitamente com aplicações de Visão Computacional, que se caracterizam por lidar com recursos pesados, como *datasets* de imagens e vídeos e computações extensivas, nas quais comumente são realizadas operações em imagens de alta resolução. A Construção de Mosaicos é uma técnica de Visão Computacional na qual um conjunto de imagens que compartilham áreas redundantes é agrupado em uma só imagem. Neste trabalho, foi analisado o impacto causado pela paralelização do extrator de características na construção de mosaicos, bem como geradas conclusões sobre o processo de paralelização do algoritmo.

PALAVRAS-CHAVE: Paralelismo. Mosaico. Visão Computacional.

Recebido: 19 ago. 2019.
Aprovado: 01 out. 2019.

ABSTRACT

DUE TO ITS HIGH COMPUTATIONAL CAPACITY, PARALLELISM IS A TOOL THAT COMBINES PERFECTLY WITH COMPUTATIONAL VISION APPLICATIONS, WHICH ARE CHARACTERIZED BY HANDLING HEAVY RESOURCES, SUCH AS IMAGE AND VIDEO DATASETS AND EXTENSIVE COMPUTATIONS. CONSTRUCTION OF MOSAICS IS A COMPUTER VISION TECHNIQUE IN WHICH A SET OF IMAGES THAT SHARE REDUNDANT AREAS IS GROUPED INTO A SINGLE IMAGE. IN THIS WORK, WE ANALYZED THE IMPACT OF FEATURE EXTRACTOR PARALLELIZATION ON THE CONSTRUCTION OF MOSAICS, AS WELL AS CONCLUSIONS ABOUT THE ALGORITHM PARALLELIZATION PROCESS.

KEYWORDS: Parallelism. Mosaic. Computer Vision.

INTRODUÇÃO

Com o passar do desenvolvimento da computação, as CPUs (*Central Processing Units*) foram ganhando maior velocidade de processamento, possibilitando realizar cálculos e operações mais eficientes e velozes. Após sequentes evoluções, chegou-se a um limite no qual não se conseguia maior velocidade com a tecnologia conhecida. Portanto, foi necessário encontrar uma outra maneira de aumentar o poder de processamento, pois o avanço da ciência comumente requer maior capacidade computacional.

De acordo com (PACHECO, 2011), em vez de construir processadores monolíticos mais velozes e complexos, a indústria tomou a decisão de colocar vários processadores simples e completos em um único chip. Dessa forma, no local que antes existia um único processador grande e complexo responsável por todas as tarefas da máquina, passou-se a ter vários mais simples. Com essa mudança tornou-se possível dividir tarefas executadas simultaneamente, cada uma alocada em um dos processadores, de maneira paralela. À essa tecnologia deu-se o nome de paralelismo.

Pela alta capacidade computacional, o paralelismo é uma ferramenta que se combina perfeitamente com aplicações de Visão Computacional, que se caracterizam por lidar com recursos pesados, como *datasets* de imagens e vídeos e computações extensivas, nas quais comumente são realizadas operações em imagens de alta resolução.

A Construção de Mosaicos é uma técnica de Visão Computacional na qual um conjunto de imagens que compartilham áreas redundantes é agrupado em uma só imagem. Um exemplo de aplicação é visto a agricultura, pois pode-se tirar várias fotografias aéreas de uma área agrícola e então juntá-las em uma só figura, facilitando a visualização e entendimento da cena como um todo. Além disso, de acordo com (WANG, 2000), essa técnica pode ser utilizada para comprimir imagens, pois as áreas em comuns, sobrepostas, são eliminadas, removendo redundâncias e, consequentemente, diminuindo a quantidade de espaço utilizado para armazenar a informação.

MATERIAL E MÉTODOS

De maneira intuitiva, para unir duas imagens, removendo redundâncias, é necessário primeiramente identificar as características das imagens. Em seguida, comparar o que se repete em ambas, identificando regiões parecidas, para, então, unir as imagens.

Para identificar as características de uma imagem existem algoritmos conhecidos como Extratores de Características. Um Extrator de Características é um algoritmo que aplica diversas etapas de processamento em uma imagem de entrada de forma que partes da imagem que não são relevantes sejam descartadas, restando apenas pontos (X, Y) que são considerados como característicos. Ou seja, com aqueles determinados pontos é possível identificar toda a imagem.

Para comparar o que se repete nas imagens, é utilizado um *matcher*. Esse algoritmo recebe os pontos característicos de cada imagem e então os compara, selecionando aqueles que mais se parecem. Ao final, com os pontos selecionados pelo *Matcher*, é calculada a Matriz de Homografia, responsável por computar a transformação de um plano para outro. Portanto, ao obtê-la, pode-se alinhar duas imagens.

Neste trabalho, foi escolhido analisar e implementar a etapa de extração de características por conta de ser uma tarefa que consome grande tempo computacional (FENG, 2008). Sendo assim, possivelmente, beneficiada pelo uso de paralelismo.

No contexto computacional, é necessário utilizar um algoritmo que seja capaz de identificar pontos presentes na imagem 1 e relacioná-lo na imagem 2. Uma variante importante neste problema, é que as imagens podem ter sido capturas em ângulos e distâncias para os objetos diferentes.

Neste trabalho, foi utilizado o Extrator de Características SIFT (*Scale Invariant Feature Transform*). A principal característica desse extrator é ser invariante em escala e rotação, ou seja, mesmo em condições nas quais de uma imagem para outra há uma mudança de rotação e/ou distância o algoritmo consegue identificar os pontos característicos (LOWE, 2004). Para isso, o SIFT aplica operações na imagem e ao final obtém uma sequência de vetores, cada um representado um ponto (X, Y) considerado como característico daquela cena.

. Para realizar o processamento paralelo neste trabalho, foi utilizada a API OpenMP, que suporta o uso de programação paralela em memória compartilhada. Esta tecnologia é baseada no modelo *fork-join*, que após uma região sequencial do código, abre uma região paralela e, então, une todas as execuções novamente em uma região sequencial.

O processo de paralelização foi experimental, buscando situações nas quais o processamento era acelerado. Notou-se que nem sempre paralelizar uma função resulta em um aumento na velocidade. Essa situação pode acontecer por dependência entre valores/objetos, casos em que é necessário sincronismo entre as *threads* e uma delas é mais demorada, entre outros.

Os pontos de maior destaque em relação ao algoritmo, demonstrado na Figura 1, são os *loops*. Pode-se perceber que em todos os *loops* aninhados apenas um dos laços é paralelizado. Essa estratégia é utilizada para maximizar o número de *Cache Hit* durante a execução.

A memória cache é uma memória de capacidade reduzida, mas muito mais veloz que a memória principal. Durante uma execução, quando se requer um determinado valor e ele não se encontra na memória cache (*Cache Miss*), é necessário ir até à memória principal, ou até ao HD (Hard Drive), para buscá-lo. Entretanto, não é eficiente realizar todo esse percurso custoso para buscar apenas uma variável inteira, por exemplo. Por esse motivo, na verdade é levado um bloco inteiro de memória até à cache, seguindo o principio de localidade espacial (HENNESY; PATTERSON, 2008).

Quando se utiliza a anotação de paralelização de um *loop for* no OpenMP, o que acontece é que uma porção daquele vetor e/ou matriz é entregue a cada thread. Dessa forma, se os laços alinhados recebem a indicação de paralelização cada *thread* ficaria com uma porção pequena do vetor/matriz para trabalhar, aumentando a quantidade de *Cache Miss* e, consequentemente, levando mais tempo para realizer o processamento.

A função responsável por realizar o cálculo da Diferença de Gaussianas não foi pararelizada, pois cada chamada lida com uma pequena quantidade de valores (27 indíces). Portanto, o tempo que cada *thread* gasta para iniciar, realizar o processo e então sincronizar é maior do que simplesmente executar de maneira sequencial. A cache também é afetada, como nos *loops*.

Figura 1 – Pseudódigo do SIFT paralelizado

Algoritmo 2 SIFT Paralelo

```
1:  para i ← 0 até quantidadeImagens faça
2:      img ← Imagens[i]
3:
4:      #Paralelo
5:      para i ← o até img.linhas faça
6:          para j ← o até img.colunas faça
7:              img[i][j] ← gaussianValue(img, i, j)
8:          fim para
9:      fim para
10:
11:     calculaDoG()
12:
13:     para i ← 0 até quantidadeImagensScaleSpace faça
14:         img ← Imagens[i]
15:
16:         #Paralelo
17:         para i ← 0 até img.linhas faça
18:             para j ← o até img.colunas faça
19:                 valorPixel ← img[i][j]
20:                 se naoExtremo()||baixoContraste() então
21:                     removePonto()
22:                 fim se
23:             fim para
24:         fim para
25:
26:         #Paralelo
27:         para i ← 0 até img.linhas faça
28:             para j ← o até img.colunas faça
29:                 valorPixel ← img[i][j]
30:                 se naoExtremo()||baixoContraste() então
31:                     removePonto()
32:                 fim se
33:             fim para
34:         fim para
35:
36:         #Paralelo
37:         para i ← o até quantidadePontosChaves faça
38:             descritor ← geraDescritor(pontosChaves[i])
39:             Descritores.insere(descritor)
40:         fim para
41:     fim para
```

RESULTADOS E DISCUSSÕES

Para realizar os experimentos do algoritmo implementado, foi utilizada a estrutura do Laborátorio de Computação Aplicada (LaCA), localizado no campus de Ponta Grossa. A configuração da máquina conta com um processador Intel(R Core(TM) i7-8700 CPU @3.20GHz, 16 GB de RAM e 12 núcleos.

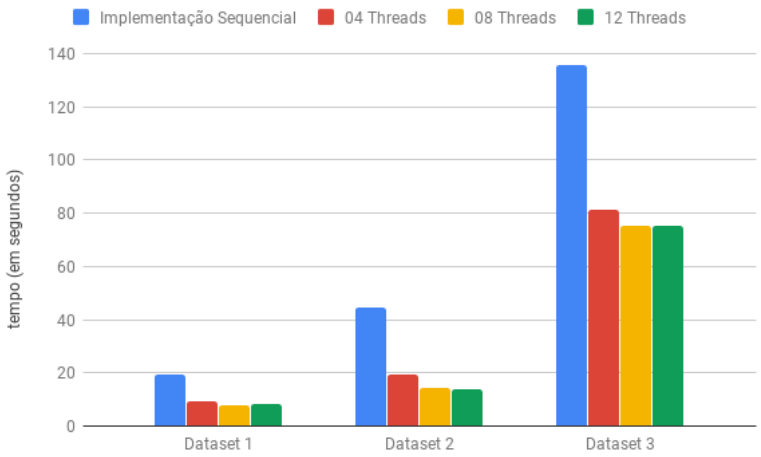
Para execução dos testes foram utilizados 3 *datasets*, de forma que fosse possível analisar o comportamento da paralelização do Extrator de Características para quantidades variadas de imagens. Os datasets contém 2, 6 e 11 imagens, respectivamente. Em cada um, foi realizada a execução do método de construção de mosaicos utilizando a implementação sequencial e a implementação paralela com 4, 8 e 12 *threads*. Todas as imagens foram processadas para escala de cinza, de modo que facilite a utlização das mesmas. Por conta do limite de páginas, apenas um dataset é demonstrado, frizando que a qualidade da imagem resultante não é influenciada pela quantidade de threads.

Figura 2 – Mosaico do Dataset 1

Como pode-se visualizar na Figura 3, há um grande aumento de desempenho ao paralelizar o extrator de características. Entretanto, os experimentos realizados demonstram que aumentar a quantidade de *threads* não significa necessariamente aumentar o desempenho, como pode-se observar comparando o desempenho com 8 e 12 threads. Esse comportamento ocorre pois, apesar do OpenMP abstrair isso do programador, gerar o código paralelo introduz uma série de operações para criação e manutenção das *threads*, o que resulta em tempo consumido para tal. Desse modo, avançar de 8 para 12 *threads* não apresentou melhorias significativas de tempo para executar o algoritmo.

Nos experimentos realizados neste estudo, foi constatado que o melhor caso para as imagens obtidas é utilizar 8 *threads* para o SIFT na construção de mosaicos.

Figura 3 – Gráfico com os resultados obtidos nos experimentos



CONCLUSÕES

A Com o desenvolvimento deste trabalho foi possível constatar que o Extrator de Característica tem um grande impacto no processo de construção de um mosaico e que, neste contexto, a programação paralela resulta em um grande aumento de desempenho, propiciando resultados mais rápidos e, consequentemente, maior facilidade em executar a construção para grande quantidade de imagens, visto que não será necessário esperar um tempo exaustivo para visualizar o resultado.

Além disso, observou-se que a *API* OpenMP ajuda o desenvolvedor a ser mais produtivo, realizando o trabalho de manutenção das *threads* de maneira automática. Entretanto, é necessário ter cuidado ao analisar e implementar o paralelismo, visto que em muitos casos, como os explicados durante o desenvolvimento deste trabalho, paralelizar mais trechos de código pode resultar em um desempenho pior. Portanto, conclui-se que paralelizar um algoritmo é uma tarefa que requer um grande tempo de análise do problema, antes de realizar a codificação.

REFERÊNCIAS

FENG, H.; et al. **Parallelization and Characterization of SIFT on Multi-Core Systems.** IEEE International Symposium on Workload Characterization, 2008, p 14-23

HENNESSY, J. L.; PATTERSON, D. A**. Arquitetura de computadores: uma abordagem quantitativa.** 5. ed. Rio de Janeiro: Elsevier, 2008.

LOWE, D. G. **Distinctive Image Features from Scale-Invariant Keypoints**. Int. J. Comput. Vision 60, 2004, p 91-110.

PACHECO, P. **An Introduction to Parallel Programming.** 1. ed. Burlington, Estados Unidos da América: Elsevier, 2011.

WANG, H. **Parallel Algorithms for Image and Video Mosaic Based Applications**. 2000. Tese (Mestrado)– Guangzhou Jinan University. Guangzhou, 2000.