MULTITHREADED FULLSEARCH: UMA IMPLEMENTAÇÃO MULTITHREAD DO ALGORITMO FULLSEARCH PARA VÍDEOS HD UTILIZANDO GPGPU

ATAIDES, Vítor Alano; GNUTZMANN, Pamela Polnow;
Pet Computação
Universidade Federal de Pelotas

AGOSTINI, Luciano Volcan; PILLA, Maurício Lima; MATTOS, Julio Carlos Balzano;
Pet Computação
Universidade Federal de Pelotas

1 INTRODUÇÃO

Atualmente, observa-se uma estagnação do crescimento do desempenho dos processadores de uso geral, devido a limitações como latência de memória, paralelismo em nível de instrução e dissipação de energia. Essas barreiras são contornadas com a inclusão de mais processadores em um único chip, os chamados multicores (ASANOVIC, 2009) .

As GPUs (*Graphics Processing Units*) exploram paralelismo há bastante tempo, usando uma arquitetura SIMD *(Single Instruction Muliple Data*), hoje também denominada SIMT (*Single Instruction Multiple Threads*). Uma GPU, hoje em dia, pode ter mais do que 30 processadores e cada um executa até 32 *threads* (processos leves) ao mesmo tempo (NVIDIA Corporation, 2008).

GPGPU (*General-purpose on Graphics Processing Units*) é a técnica de usar a GPU, normalmente usada para processamento de computação gráfica, para realizar a computação de aplicações que normalmente é feita pelo processador principal do computador ou CPU. O uso da GPU pode ser bastante eficiente, se usado devidamente, pelo seu grande número de processadores. Conforme pode ser observado na Figura 1, esse grande número de *threads* faz com que o desempenho das placas gráficas ultrapasse em muito o desempenho das CPUs para operações de ponto flutuante com precisão simples (NVIDIA Corporation, 2008).

Este trabalho tem por objetivo utilizar o alto poder de paralelismo da GPU para executar parte do algoritmo do módulo de Estimação de Movimento (ME) do codificador de vídeo do padrão H.264. Mais especificamente, foi implementado o algoritmo de estimação de movimento *Full Search* (FS) (KUHN, 1999).

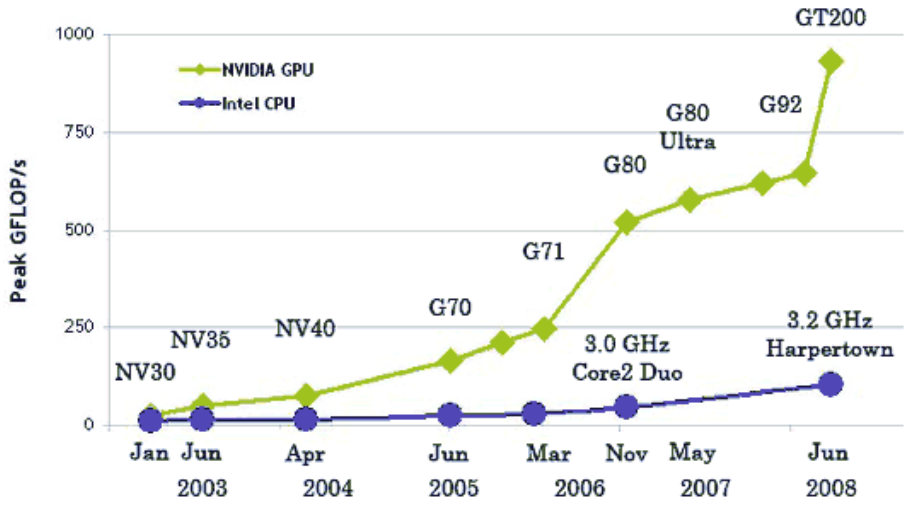

Figura 1 - Comparativo de desempenho máximo entre CPUs e GPUs (NVIDIA, 2009)

O *FullSearch* é um algoritmo de busca muito utilizado na codificação de vídeos, mais precisamente no módulo de estimação de movimento. Apesar de ser o algoritmo mais eficiente na busca por resultados ótimos, é também o que possui maior complexidade computacional. A estimação de movimento consiste na comparação entre dois *frames* (quadros) de um vídeo em busca de redundâncias temporais que permitam diminuir a quantidade de informação a ser codificada. Na ME, o quadro atual é dividido em blocos. Cada bloco do quadro atual é procurado dentro de uma área de busca própria no quadro de referência e um vetor de movimento é gerado indicando a posição do bloco de maior similaridade da área de busca. Um vídeo *full* HD (*High Definition*) possui a resolução de 1920x1080 pixels. Usando blocos de 4x4 pixels e considerando uma área de busca de 32x32 pixels existe um total de 841 blocos candidatos por área de busca. Realizar estes cálculos para todos os blocos do vídeo HD de forma sequencial é praticamente inviável quando se pretende alcançar desempenho para codificação em tempo real (30 quadros por segundo).

2 METODOLOGIA

Ao estudar o algoritmo FS, foi perceptível o grande potencial de paralelização deste algoritmo, visto que os cálculos de similaridade entre blocos são independentes entre si. Paralelizar o algoritmo diminui linearmente seu tempo de execução. Com grandes níveis de paralelismo, o algoritmo FS pode atingir taxas de processamento similares a algoritmos de busca menos complexos, mantendo o resultado ótimo em termos de qualidade. Para paralelizar o algoritmo usando a GPU, foi estudada a arquitetura CUDA (*Compute Unified Device Architecture*), que é a arquitetura das novas placas de vídeo da NVIDIA (NVIDIA Corporation, 2008). Além da arquitetura, ainda existe a biblioteca CUDA para a linguagem de programação C, que facilita a construção de programas na placa gráfica. Após esses estudos, foi decidido implementar o algoritmo FS utilizando uma GPU da NVIDIA e a biblioteca CUDA. O software desenvolvido recebeu o nome de *Multithread FullSearch*.

No *Multithread FullSearch*, são criadas diversas *threads*, cada uma ficando responsável pelo cálculo do vetor de movimento de um bloco. Cada uma destas *threads* retorna um vetor de movimento (x,y), e todos são colocados em uma matriz de vetores, que é a saída do algoritmo, exatamente como ocorria no *Full Search* sequencial.

O programa desenvolvido foi dividido em 3 módulos: *Main*, *LoadVideo* e *FullSearch*, porém, apenas no último é usado paralelismo. Na Figura 2, é apresentada um visão geral do pipeline desse algoritmo.

Figura 2 – Overview do programa

O módulo *Main* funciona como um gerente de todo o programa. Sua principal função é a alocação de memória, tanto de memória principal do computador quanto da memória da placa de vídeo. Além disso, é o módulo *Main* que chama as demais funções. Após alocar a memória na placa gráfica o módulo *Main* chama a função *LoadVideo*. Usa-se a saída de *LoadVideo* como parâmetro da *FullSearch*. É na função Main que se definem quantas *threads* serão usadas na GPU.

O objetivo do *LoadVideo* é carregar apenas a parte de luminância, que representa o vídeo em escala de cinza, de um vídeo *Full* HD. Para isso, esse módulo faz uso de uma matriz tridimensional, onde a primeira dimensão é a altura do quadro, a segunda é a largura do quadro e a terceira é a sequencia de quadros do vídeo. Para carregar apenas a luminância, o módulo *LoadVideo* simplesmente pula a parte de crominância - que representa as cores - durante a leitura do vídeo. Em um vídeo *Full* HD, cada quadro é representado por todos os bytes de luminância seguido pelos bytes de crominância, onde a quantidade de bytes da primeira é o dobro dos bytes da segunda (PEARLMAN, 2009).

O módulo *FullSearch*, como o nome sugere, realiza a busca utilizando o algoritmo *Full Search*. Sua implementação utilizando *threads* não difere muito da forma sequencial. Esse módulo é executado por todas as *threads* paralelamente. A quantidade de *threads* depende da quantidade de blocos em que os quadros serão divididos e cada *thread* ficará responsável por um bloco, buscando o melhor casamento de bloco em sua área de busca e retornando um vetor de movimento a partir do resultado da busca.

3 RESULTADOS E DISCUSSÕES

Os resultados obtidos com a execução do programa podem ser observados na Tabela 1. O programa foi executado com 5 vídeos diferentes como entrada, cada um com 200 quadros. Os quadros foram divididos em blocos de 4x4 amostras e a área de busca foi definida em 40x40 amostras e, no total, foram usadas 841 *threads*, uma para cada bloco. O tempo de execução foi monitorado. Podem ser observados nas colunas “Seg./quadro” uma média do tempo de execução para cada quadro do vídeo, considerando a versão GPU e a versão sequencial. Os testes utilizando versões sequenciais do *Full Search* utilizaram as mesmas configurações utilizadas para o teste com GPU É notável a diferença no

tempo de execução por quadro: o ganho de velocidade da versão GPU é de quase 9,5 vezes. Os testes foram repetidos 10 vezes para cada vídeo.

Tabela 1 – Avaliação de desempenho

Vídeo	Seg./quadro (Sequencial)	Seg./quadro (GPU)
Man in the car	59,716	6,063
Blue sky	59,549	6,077
Pedestrian area	55,290	6,067
Rolling tomatoes	58,673	6,080
Rush hour	57,075	6,063

4 CONCLUSÕES

A implementação do algoritmo *Full Search* em GPU foi concretizada e o objetivo de demonstrar as potencialidades da exploração do paralelismo para estimação de movimento foi alcançado. Os resultados indicam uma aceleração de 9,5 vezes na execução da ME com a solução desenvolvida neste trabalho, o que é um ganho muito expressivo.

Como trabalho futuro, tem-se a idéia de executar testes e comparações do programa criado com uma versão do *Full Search* que execute em multicores. É esperado, também, paralelizar mais partes do módulo de Estimação de Movimento, para que se possa fazer com que a codificação de vídeos HD se torne um processo mais rápido. Nessa mesma linha, cabe a implementação de outros algoritmos de Estimação de Movimento e ainda de outros módulos do codificador de vídeo.

5 REFERÊNCIAS

NVIDIA CORPORATION. **NVIDIA CUDA: Programming Guide Version 3.0**. Santa Clara, EUA: NVIDIA Corporation, 2008. 98 p.

KUHN, P. Algorithms, **Complexity Analysis and VLSI Architecture for MPEG-4 Motion Estimation**. Boston: Kluwer Academic Publishers, 1999.

HENNESY, J. L.; PATTERSON, D. A. **Computer architecture, fourth edition: a quantitative approach**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

ASANOVIC, K.; BODIK, R.; DEMMEL, J.; KEAVENY, T.; KEUTZER, K.; KUBIATOWICZ, J.; MORGAN, N.; PATTERSSON, D.; SEN, K.; WAWZRYNEK, J.; WESSEL, D.; YELICK, K. **A view of the parallel computing landscape**. Communications of the ACM, 52(10):56-67, 2009.

PEARLMAN, W.A.; ISLAM, A.; NAGARAJ, N.; SAID, A.; **Efficient, low-complexity image coding with a set-partitioning embedded block coder.** Comput. & Syst. Eng. Dept., Rensselaer Polytech. Inst., Troy, NY, EUA. Circuits and Systems for Video Technology, 1219–1235, 2009.