Multiplicação de matrizes

Bernardo Prina Righi
Universidade do Vale do Taquari
Lajeado, Brasil
bernardo.righi@universo.univates.br

Douglas Somensi
Universidade do Vale do Taquari
Lajeado, Brasil
dsomensi@universo.univates.br

Abstract - Problem solving using arrays is widely used and so it is of paramount importance in computing. The use of matrices can be observed in areas such as computer graphics, graphs and machine learning. In this way, to operate arrays efficiently is very important for the performance of algorithms.

One of the most used matrix operations is multiplication. As the matrices to be multiplied increase, the computational performance of the algorithms may become a hindrance.

Because of this, parallel computing has become a standard solution to address such a problem. In this paper we present a comparison between the sequential and parallel approaches for multiplication of matrices using CUDA and OpenMP. The result of the analysis performed between matrix size and multiplication performance shows the importance of parallelization mainly for high order matrices.

Palavras-chave - GPU, CUDA, OpenMP, Computação Paralela, Multiplicação de Matrizes

INTRODUÇÃO

Um dos fundamentos da computação científica é a operação com matrizes, sendo uma das principais operações a multiplicação. Esta operação é bastante comum em algoritmos de diversas áreas e possui um grande potencial de consumo de tempo computacional.

Uma abordagem padrão para otimização de processamento de dados é o uso de paralelismo computacional e uma forma eficiente e massivamente utilizada de paralelização é o uso de uma Unidade de Processamento Gráfico (Graphics Processing Unit - GPU).

Uma outra maneira simples de se obter paralelismo utilizando apenas os núcleos de uma CPU é utilizando OpenMP, uma interface de programação paralela de memória compartilhada para arquitetura de múltiplos processadores. Sendo assim, neste trabalho a computação paralela utilizando GPU e OpenMP é realizada para computar a multiplicação de matrizes, pois sendo uma forma mais otimizada do que a forma sequencial do CPU a comparação com a GPU fica mais interessante do ponto de vista de performance.

CONCEITOS BÁSICOS DE GPUS E PLATAFORMA CUDA

GPUs são unidades de processamento especializadas em processamento gráfico da classe SIMT (Single Instruction Multiple Threads), neste modelo, múltiplas threads independentes são executadas de maneira concorrente utilizando uma instrução única.

Desde 2006, com lançamento da plataforma CUDA, GPUs vêm sendo utilizadas com propósito geral de computação. Uma maneira simples de compreender a diferença entre uma GPU e uma CPU é comparar o modo que as mesmas processam suas tarefas. Uma CPU possui alguns núcleos otimizados para o processamento serial sequencial, enquanto uma GPU tem uma arquitetura paralela gigantesca que consiste em milhares de núcleos menores e eficientes, criados para lidar com múltiplas tarefas simultaneamente.

De maneira geral, GPUs da NVIDIA são divididas em SMs (Stream Multiprocessors), local onde um conjunto de threads, denominado warp, é executado. Cada SM possui vários núcleos de processamento chamados de CUDA cores (ou Streaming Processor - SP), que por sua vez possuem pipelines completos de operações aritméticas (ALU) e pontos flutuantes (Floating Point Unit - FPU). Cada SM possui uma memória dedicada chamada de memória compartilhada (shared memory).

Essa memória é fisicamente próxima dos cores e possuem acesso muito rápido, porém são pequenas (na ordem de KB). Além disso uma SM possui um cache de instruções, um cache de constantes, uma unidade de funções especiais (SFU - Special Function Units) e um bloco para escalonar warps (Multithread Instruction Fetch and Issue Unit - MT Issue). O número de SMs, cores, tamanho de cache, dentre outros, podem variar de acordo com o modelo da GPU.

Threads podem ser agrupadas em blocos que executam na mesma SM compartilhando a mesma shared memory. Cada SM possui um limite máximo de blocos, que por sua vez possui um limite máximo de threads. Um conjunto de blocos criados por um código CUDA é chamado de grid. Os valores dos limites também são parâmetros de cada modelo de GPU.

CUDA é a plataforma que permite utilizar uma GPU com propósito geral de computação. É um software desenvolvido pela NVIDIA corporation, portanto somente GPUs da NVIDIA são capaz de executar códigos CUDA. O desenvolvimento de códigos em CUDA podem ser realizados utilizando C/C++ juntamente com alguns comandos específicos da plataforma.

MULTIPLICANDO MATRIZES NA GPU

A primeira abordagem para paralelização da multiplicação de matrizes utilizando uma GPU é disparar diversas threads fazendo com que cada uma calcule um elemento da matriz resultante C. Nesta abordagem, cada thread lê uma linha de A e uma coluna de B para computar o elemento cij de C, sendo $i=1\cdots m$ e $j=1\cdots w$.

Utilizando essa abordagem de multiplicação, as matrizes A e B são carregadas na memória global w e m vezes, respectivamente. Com isso o algoritmo faz muitos acessos à memória global, que possui alta latência e baixo throughput.

Com intuito de extrair a eficiência máxima que uma GPU pode entregar, a segunda abordagem de multiplicação de matrizes na GPU tem como objetivo utilizar a memória compartilhada de cada bloco de threads visando reduzir o número de acessos a memória global do dispositivo, para isso as matrizes são subdivididas em pequenos blocos.

Essa técnica é conhecida como multiplicação por ladrilhamento e diferentemente da abordagem anterior, no qual toda linha de A e coluna B era multiplicada de uma só vez gerando assim um elemento de C, no ladrilhamento tem-se submatrizes de A e B que vão gerar um valor parcial de um elemento de C. Quando todos os ladrilhos forem processados o valor final de cada um dos elementos da matriz C será o somatório de cada elemento parcial obtido pela multiplicação de cada uma das linhas e colunas sub matrizes A e B, respectivamente.

O objetivo principal de se usar o ladrilhamento em GPU é carregar cada submatriz na memória compartilhada do bloco de threads. O caso ideal seria alocar toda matriz dentro de uma memória compartilhada. Porém, como já mencionado, essa memória é de tamanho reduzido quando comparada com a memória global.

Dessa forma, a ideia é carregar na memória compartilhada apenas submatrizes de A e B, na qual cada bloco de threads pode compartilhar seus dados de maneira rápida. Com isso, um requisito de projeto é que o ladrilho caiba dentro de um bloco de threads da GPU. O valor do tamanho do bloco varia de acordo com o dispositivo e cabe ao programador conhecer a arquitetura para melhor aproveitá-la.

MULTIPLICANDO MATRIZES NA CPU COM OPENMP

A ideia de ladrilhamento das matrizes também é aplicada na CPU. Na GPU, as threads são disparadas e com os índices de cada uma delas é possível operar sobre os dados de acordo com seus blocos. Na CPU esses índices são obtidos por loops de controle.

O objetivo principal do ladrilhamento também é o mesmo: tirar proveito da arquitetura de CPU através da memória cache de acesso rápido. Mesmo no código sequencial já se obtém um ganho de performance.

Para realizar paralelização por meio da CPU é utilizado a OpenMP. A API dá suporte às linguagens C/C++ e Fortran, e basicamente o que deve ser incluído ao código original são diretiva em trechos de códigos que devem ser paralelizado. Com isso, a API é capaz de disparar threads utilizando os núcleos da CPU em questão.

TESTES

Foram realizados testes para comparar a performance da multiplicação de matrizes utilizando a computação sequencial e a paralela, utilizando CUDA e OpenMP. Os testes foram executados em uma máquina com sistema operacional Ubuntu 18.04, com processador Intel Core i5, 3.70GHz, Cache 9MB e 16GB de memória RAM e uma placa gráfica NVIDIA Geforce GTX 1070Ti. Os testes foram realizados da seguinte forma:

- A multiplicação foi realizada considerando a implementação por ladrilho sequencial, paralelizada em OpenMP e em GPU.
- A multiplicação foi executada em três diferentes configurações: 1 vez, 100 vezes, e 1000 vezes. As duas últimas paralelizadas com OpenMP.
- As matrizes e os ladrilhos foram ajustados para máximo desempenho.
- O desempenho foi medido em termos de tempo de execução, speedup, através da lei de Amdahl, e em GFLOPS.

A lei de Amdahl, é usada para encontrar a máxima melhora esperada para um sistema em geral quando apenas uma única parte do mesmo é melhorada. Isto é frequentemente usado em computação paralela para prever o máximo speedup teórico usando múltiplos processadores.

O speedup de um programa usando múltiplos processadores em computação paralela é limitado pelo tempo necessário para a fração sequencial de um programa. Por exemplo, se o programa precisa de 20 horas usando um único núcleo de processamento, e a parte específica de um programa que demora uma hora para executar não pode ser paralelizado, enquanto as 19 horas restantes (95%) do tempo da execução pode ser paralelizado, independente de quantos processadores são dedicados a execução paralela deste programa, o tempo de execução mínima não pode ser menor que aquela crítica uma hora. Por isso o aumento de velocidade é limitado em no máximo 20x.

CARACTERÍSTICAS DA GPU UTILIZADA

A GPU GP104 da GTX 1070Ti é composta por 4 GPCs, 19 multiprocessadores streaming Pascal e 8 controladores de memória. Cada cluster de processamento gráfico apresenta uma engine de rasterização dedicada e cinco multiprocessadores streaming.

A arquitetura se desenvolve através da distribuição dos CUDA cores nos multiprocessadores streaming. Cada SM apresenta 128 núcleos CUDA, 256 KB de capacidade de arquivo de registro, unidade de memória compartilhada de 96 KB, 48 KB de memória cache L1 e 8 unidades de textura.

MULTIPLICAÇÃO PARA PRECISÃO SIMPLES

Na realização das multiplicações de matrizes foi optado pela precisão simples(float). Para obter eficiência máxima, as matrizes escolhidas possuem número de linhas e colunas divisíveis pelo tamanho máximo de ladrilho, ou seja, um bloco de 32×32. Dessa forma um grid de blocos se encaixa perfeitamente nas dimensões da matriz, sem necessidade de verificações na função de kernel da GPU, o que ocorreria se o ladrilho não fosse divisível.

As ordens das matrizes escolhidas, tanto de A quanto de B, variam de 32×32 até 2048×2048. Com esses valores, as matrizes cabem dentro da memória global.

CPU + OPEN MP × GPU

O desempenho do teste executado apenas uma vez na primeira configuração de matriz, a GPU possui uma taxa de GFLOPS cerca de 30x maior do que a CPU+OpenMP.

A partir desta configuração a proporção dispara para mais de 200x, deixando claro que a GPU efetua muito mais cálculos por segundo do que a CPU e a CPU+OpenMP. A diferença do tempo de execução até a multiplicação de matrizes de 640×640, não é tão perceptível. Após a configuração 1024×1024 o tempo de execução começa a fazer bastante diferença, sendo que para última configuração a GPU executa a multiplicação em torno de 1 seg e a CPU+OpenMP mais de 50 seg.

Por fim, o speedup, na qual é possível observar a superioridade da GPU para com a CPU+OpenMP, principalmente quando a ordem das matrizes aumentam.

No desempenho dos testes executando a multiplicação 100 e 1000 vezes, é possível observar que a diferença de GFLOPS se mantém. Porém, no caso da GPU, o valor aumenta um pouco para as matrizes menores. No caso da ordem 32×32 ocorre um acréscimo de quase 3x. As diferenças de tempo de execução também se mantém, com uma pequena oscilação.

Com isso o valor do speedup também oscila pouco, sendo o valor mínimo em torno de 25 e o máximo por volta de 325.

De maneira geral, é possível notar a diferença de desempenho da GPU para CPU+OpenMP. A capacidade de processamento é extremamente maior sendo muito vantajoso seu uso, principalmente quando a multiplicação de matrizes atinge a ordem de 1024×1024, no qual o crescimento da curva de tempo de execução para CPU+OpenMP é muito alto.

CONCLUSÃO

Neste trabalho foi realizado uma comparação entre multiplicação de matrizes utilizando computação paralela e sequencial. A paralelização dos códigos foi alcançada utilizando CUDA, plataforma de codificação em GPU, e OpenMP, framework de paralelização em CPU.

Para realizar a comparação das abordagens, foram elaborados diversos testes utilizando diferentes tamanhos de matrizes. Resumidamente, os resultados dos testes mostraram a importância da paralelização ao multiplicar matrizes, principalmente quando o número de elementos das matrizes utilizadas ultrapassa a ordem de 1 milhão.

Para casos como este, a GPU se mostrou centenas de vezes mais rápida do que a CPU + OpenMP. Por fim, ficou claro que para aplicações que utilizam matrizes de grande porte, paralelizar os processos utilizando GPU é uma forma extremamente eficiente de aumentar a performance final.

REFERÊNCIAS

NVIDIA CUDA C programming guide. NVIDIA Corporation, PG-02829-001_v10.1. Disponível em: https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

Blaise Barney, Lawrence Livermore National Laboratory. Introduction to Parallel Computing, UCRL-MI-133316. Disponível em: https://computing.llnl.gov/tutorials/parallel_comp/

Lei de Amdahl, https://pt.wikipedia.org/wiki/Lei_de_Amdahl, Acessado 13/06/2019