Trabalho apresentado no XXXVII CNMAC, S.J. dos Campos - SP, 2017.

Proceeding Series of the Brazilian Society of Computational and Applied Mathematics

Aplicação do Gradiente Conjugado Utilizando CUDA

Renan Gabriel de S. Pegaiane[1]
Fábio R. Chavarette[2]
Maria Gabriella R. dos R. Pegaiane[3]
Departamento de Engenharia Mecânica, UNESP, Ilha Solteira, SP

1 Introdução

Muitos problemas em diversas áreas da engenharia recaem em grandes sistemas lineares, e para resolver esses sistemas existem os métodos diretos e os iterativos, cada um com sua particularidade de aplicação. Os métodos iterativos apresentam grande superioridade com relação ao método de solução direta em determinadas aplicações, por exemplo, em casos que a matriz de coeficientes é uma matriz esparsa, ele é um método mais econômico pelo fato de utilizar menos memória do computador. As vantagens desse tipo de método é que ele se auto corrigi caso um erro é cometido, podem ainda ser usados para arredondamento de soluções obtido por métodos exatos e em determinadas situações podem também ser utilizados para resolver equações não lineares [2].

Nesse trabalho é empregado um método iterativo para resolver sistemas lineares, o método dos gradientes conjugados, que será desenvolvido de forma tradicional na linguagem C, ou seja, o programa é executado na CPU da máquina e posteriormente o mesmo método será desenvolvido utilizando uma linguagem paralela em Cuda que é executada na GPU da máquina para comparação do tempo de consumo computacional envolvido em ambas as linguagens de programação.

2 CUDA (*Compute Unified Device Architecture*)

O grande desafio em programação é fragmentar os processos a modo que eles possam ser executados em concorrência. CUDA foi desenvolvido para facilitar a programação paralela, sua curva de aprendizado é relativamente baixa, devido sua base de desenvolvimento ser oriundas de linguagens padrões (C, C++ e FORTRAN) [3].

CUDA Possui três abstrações (hierarquia de grupos, hierarquia de threads e memória compartilhada) cujo são expostas as linguagens padrões por meio de extensões de linguagem. Essas abstrações possibilita o paralelismo de pequenos ou grandes trechos de código

[1]renanpegaiane@hotmail.com
[2]chavarette@gmail.com
[3]gabriella_maria@hotmail.com

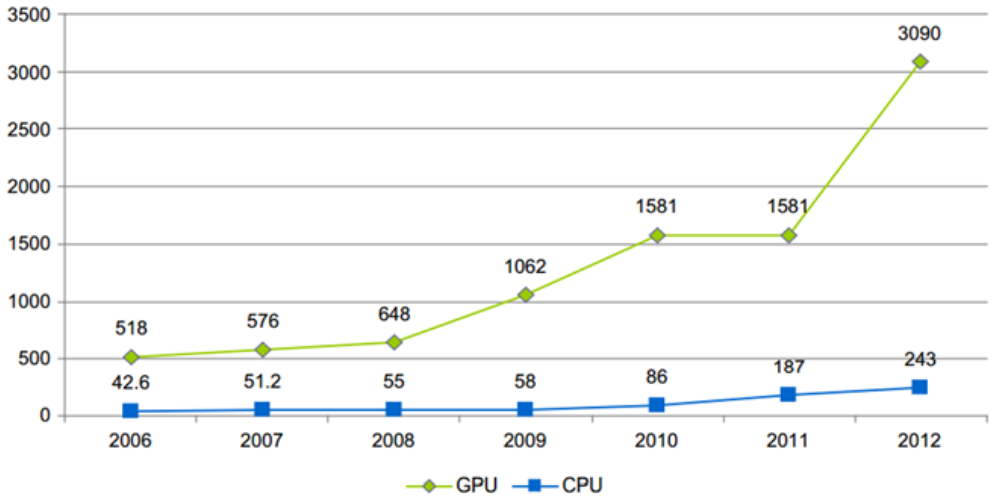

Figura 1: Comparação entre GPU e CPU em gigaflop. Fonte: Cook, Shane (2013, p. 13)

(granularidade fina ou graça respectivamente), orientando o programador a dividir problemas em sub-problemas que podem ser solucionados independentemente por blocos de threads e cada sub-programa pode ser dividido em granularidades ainda menores onde serão executados independentemente dentro da thread de cada bloco.

A distribuição de tarefas é feita de maneira automática, os blocos de thread são reservados em qualquer uma das matrizes de multiprocessadores (SMs) da GPU de maneira concorrente ou sequencial. Desta maneira o programa CUDA compilado pode executar em qualquer número de multiprocessadores, pois em apenas em tempo de execução é necessário saber a contagem de multiprocessadores físicos

A figura 1 apresenta a comparação do desempenho em gigaflop entre GPUs (placas gráficas fabricante NVIDEA) e CPUs (processadores fabricante Intel) é possível notar que a partir de 2009 houve uma divergência significante entre eles, onde a GPU quebra a barreira de 1 teraflop (1000 gigaflop) [1].

A razão da discrepância na capacidade entre a CPU e a GPU, é que a arquitetura da GPU foi desenvolvida exclusivamente para processamentos paralelos. Seus transmissores são dedicados a processamento de dados e não tratamento de dados e controle [3].

Referências

[1] S. Cook. *CUDA Programming: A Developers Guide to Parallel Computing with GPUs*. Elsevier, 2013

[2] N. M. F. Bertoldi. *Cálculo Numérico*. Prentice Hall, 2006

[3] Nvidia. *CUDA C Programming Guide*. Nvidia, 2014

[4] http:www.nvidia.comobjectcuda_home_new.html. Acessado em 19/12/2016