Cpu Gpu Language

Abstract
The Linguagem Cpu Gpu was created for improving the development of aplication that use GPU in your algorithm. With this, was developed a high level language that abstract specific features of video card allowing applications can run on any machine with or without GPU.

1 Introdução

Os projetos tradicionais utilizam-se de algoritmos para processar informações para as suas aplicações. Esses algoritmos foram elaborados para serem processados pela CPU com as suas características de acesso a memória primária. Alguns algoritmos podem ser modificados para que possam ser processados pela GPU, ao invés da CPU. A vantagem é a liberação da CPU para outras tarefas mais importante.

Além da liberação da CPU, transferi os algoritmos para a GPU pode possibiltar um ganho de desempenho em função do número de núcleos de processamento que uma GPU pode ter. Mas para isso, será precisa paralelizar o algoritmo para que ele possa ser executado em diferentes núcleos.

Muitas das vezes, paralelizar o algoritmo pode ser uma tarefa até impossível, devido as suas características de implementação. Por tanto, nem todos os algoritmos podem ser executados na GPU, pensando em ganho de desempenho.

Em alguns casos, é possível paralelizar o algoritmo e não conseguir um ganho de desempenho com o uso da GPU em função das características de como a GPU funciona. Então, é importante conhecer como os algoritmos serão executados na GPU para que possa entender porque o ganho de desempenho não foi tão significativo ou pior.

Na parte 2 irá abordar os aspectos de placa de video programável e suas características. A parte 3 irá comentar sobre a origem da criação da Linguagem Cpu Gpu parte 4 irá comentar as características básicas da linguagem. A parte 5 irá comentar as características da máquina virtual que a linguagem gera. Por último, a parte 6 irá comentar sobre o recurso de Just in Time implementado pela Linguagem Cpu Gpu

2 Gpu

As placas de video foram desenvolvidas para processar algoritmos sobre programação gráfica e enviar o resultado para o monitor. Atualmente, as placas de video programaveis estão sendo utilizadas também para processar algoritmos que diz respeito a aplicação.

Com a vinda dessas placas programáveis, está sendo possível transferir algoritmos que antes eram executados na CPU para a GPU. Esses algoritmos, quando devidamente escrito, tem um tempo de resposta muito superior a CPU porque o número de núcleos de processamento que uma GPU possui é muito superior a CPU, dependendo da sua configuração.

Diversas fabricantes de placa de video estão criando APIs de desenvolvimento para a placa de video programável. A NVidia desenvolveu o CUDA para permitir o envio de algoritmos para a placa gráfica.

Atualmente, para desenvolver para as placas da NVidia, é precisa ter CUDA instalado na máquina a qual será executado a aplicação e as bibliotecas para o ambiente de desenvolvimento.

Além de todos os requisitos de ambiente de desenvolvimento e de execução que CUDA necessita, é importante conhecer em qual placa gráfica o algoritmo será executado. Isso porque o código a ser enviado para GPU é compilado para um conjunto de placa gráficas, sendo necessário recompilar toda vez que for tracado a placa gráfica.

3 Linguagem Cpu Gpu

Como foi comentado na sessão anterior, para desenvolver uma aplicação para a GPU, é preciso preparar o ambiente de desenvolvimento, de execução e conhecer a placa que está sendo executada.

A Linguagem Cpu Gpu foi criado com o objetivo de abstrair esses requisitos de dependência, tanto do ambiente de desenvolvimento, quando da execução e até mesmo da placa de video.

Essa Linguagem irá considerar um ambiente com uma GPU genérica, abstraindo os requisitos de placa de video e de seus recursos, emulando caso a placa de video programável não suportar. O código implementando para a GPU genérica será convertida para uma implementação específica para

a placa de video em tempo de execução e não de compilação.

Com esses requisitos, um código escrito na Linguagem Cpu Gpu poderá ser executado em diferentes máquina com diferentes placa de video sem precisar recompilar a aplicação.

4 Características da Linguagem

A Linguagem Cpu Gpu é orientado a objeto. Assim, todos os códigos fontes dela serão orientados a objeto. Parecido com a Linguagem Java, essa linguagem possui os conceitos de hierarquia de classe e de polimorfismo.

Essa Linguagem possui o recurso de Coleta de Lixo quando manualmente é solicitado ou quando há falta de memória. O Coletor de Lixo tem o papel de remover objetos da memória não mais utilizados ou objetos que ninguém mais o referência.

A linguagem tem o recurso de gerência de memória que tem como papel de não liberar os objetos coletados ou liberados a não ser que haja falta de memória. O gerenciador de memória irá segurar os objetos coletados ou liberados para que possa ser retornado quando for feito uma nova alocação de objeto da mesma classe.

5 Máquina Virtual

A Linguagem Cpu Gpu foi desenhada para ser de alto nível. Isso significa que o desenvolvedor não deva se importar em qual ambiente estará sendo executado a aplicação. Esse Linguagem especifica uma máquina virtual que irá executar a aplicação. Essa máquina virtual foi especificada com os recursos genéricos de placa de video. Quando as instruções genéricas de placa de video for executada, será convertida para instruções específicas da placa. Dessa forma, um código escrito para uma placa genérico, pode ter o mesmo desempenho de um código escrito especificamente para uma placa gráfica.

A Máquina Virtual da Linguagem Cpu Gpu possui instruções tanto de CPU quanto de GPU. Todas essas instruções são convertidas para Assembler e só depois são executadas. As instruções da GPU serão convertida para Assembler da placa de video do computador que está executando. Caso o computador não tenha uma placa de video programável, a Linguagem Cpu Gpu irá simular o processamento com a CPU aproveitando o algoritmo paralelizado em diversas Threads.

6 Just in Time

Com o objetivo de concorrer com a velocidade de execução que um programa escrito em C++ possui, a Linguagem Cpu Gpu foi criada com o recurso Just in Time .

A funcionalidade Just in Time faz com que os bytecodes da máquina virtual da Linguagem Cpu Gpu sejam todas convertida para instruções de Assembler. Assim, ao executar uma aplicação escrita na Linguagem Cpu Gpu em tempo de execução, todos os bytecodes dessa aplicação será convertida para Assembler da arquitetura da máquina que está executando e será executado. Dessa forma, o tempo de execução de um programa escrito em C++ com um escrito pelas Linguagem Cpu Gpu será parecido.

7 Conclusão

O objetivo da linguagem é criar um ambiente genérico para que o mesmo código fonte possa ser executada em diferentes placas de video. Para isso, em tempo de execução, é necessário gerar o código específico para a placa de video do usuário que está utilizando a aplicação.

A Linguagem Cpu Gpu foi criada para que o desenvolvedor possa abstrair o ambiente da ser executada e possa focar no problema da aplicação. Essa linguagem se encarregará em gerar o código para a placa de video ou simula-la, caso não tenha.

8 Referências

[1] Breder Projects. Breder Bernardo. http:// www.breder.org

[2] NVIDIA. CUDA Toolkit Documentation http:// docs.nvidia.com/ cuda

[3] NVIDIA. CUDA C Programming Guide. http:// docs.nvidia.com/ cuda/ cuda-c-programming-guide

[4] Parallel Thread Execution ISA. http:// docs.nvidia.com/ cuda/ parallel-thread-execution

[5] Inline PTX Assembly in CUDA. http:// docs.nvidia.com/ cuda/ inline-ptx-assembly

[6] NVIDIA CUDA Compiler Driver NVCC. http:// docs.nvidia.com/ cuda/ cuda-compiler-driver-nvcc

[7] Rahul Mangharam, Aminreza Abrahimi Saba. Anytime Algorithms for GPU Architectures

[8] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. NVIDIA Tesla: A Unified Graphics and Computing Archi-tecture. IEEE Micro, 2008.

[9] J. Nickolls, I. Buck, and M. Garland. Scalable Parallel Programing with CUDA, 2008.

[10] J. Grass, S. Zilberstein, and E. Moss. Anytime Database Algorithms. U. Amherst, Tech. Report., 1995.

[11] J. W. S. Liu and et. al. Algorithms for Scheduling Imprecise Computations. Computer, 24(5):58–68, 1991.