Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Caio Costa Salgado

Acelerando transporte radiativo ao redor de buracos negros com GPUs

São Paulo
Dezembro de 2017

Uso de GPGPU na Análise de Buracos Negros

Monografia final da disciplina
MAC0499 – Trabalho de Formatura Supervisionado.

Supervisor: Prof. Dr. Rodrigo Nemmen da Silva

São Paulo
Dezembro de 2017

Resumo

Aqui vai o resumo que ainda tem que ser feito....

Palavras-chave: GPGPU, CUDA, HPC, Monte Carlo, Transferência radioativa, Buraco Negro.

Abstract

And here will be the english abstract, that still need to be done....

Keywords: GPGPU, CUDA, HPC, Monte Carlo, Rasioactive Transfer, Black Hole.

Sumário

Lista de Abreviaturas

GPU	Unidade de processamento Gráfico (*Graphics Grocessing Unit*)
CPU	Unidade de Processamento Central (*Central Processing Unit*)
GPGPU	Unidade de processamento Gráfico de Propósito Geral (*General Purpose Graphics Processing Unit*)
CUDA	Computação em Arquitetura unificada de dispositivos (*Compute Unified Device Architecture*)
HCP	Computação de Alta Performance (*High Performance Computing*)
SIMD	Única Instrução Múltiplos dados (*Sigle Instruction Multiple Dada*)

Capítulo 1

Introdução

Uma grande dúvida dos astrofísicos e também de toda a comunidade científica é o que ocorre em um buraco negro e em suas proximidades. Na busca de respostas programas de computador são feitos com o intuito de simular essa região e talvez trazer alguma luz, um desses programas é o **grmonty**(Dolence *et al.*, 2009) (nome reduzido, em inglês, de Monte Carlo para Relatividade Geral).

Programas dessa natureza tendem a ser muito intensos no que diz respeito ao processamento, exigindo muito das CPUs, estas tornam-se assim um limitante, um gargalo, para a velocidade com a qual o programa pode devolver um resultado. É neste contexto que buscamos aplicar métodos de *Computação de Alta Performance* para otimizar ao máximo o uso todos os dispositivos do computador (hardware) que temos disponíveis.

Muitas das técnicas de HPC exploram a paralelização, o que pode ser feito massivamente por um hardware específico nesse caso as *unidades de processamento gráfico*, GPU. Tais dispositivos são muito populares e já presentes em muitas máquinas domésticas e até em smartphones, eles são confeccionados primordialmente para processamento gráfico em jogos digitais, porém graças aos avanços recentes tais dispositivos tem se tornado mais genéricos e respondendo a uma gama maior de problemas.

Ao analisar o funcionamento do **grmonty** - por sua característica de simulador de partículas - é possível classificar parte de sua execução no modelo SIMD (sigla em inglês para única instrução múltiplos dados), uma vez que simula a trajetória da cada fóton de maneira independente. Dada essa informação podemos explorar o poder computacional das GPUs afim de paralelizar a execução do código, aumentando drasticamente sua a performance.

Existem vários programas que simulam essas regiões próximas a buracos negros, porém poucos tem a amplitude e relacionam diferentes propriedades físicas, em especial a relatividade geral, como o **grmonty**. Tornando o programa mais performático e lhe fornecendo a capacidade de executar em computadores domésticos com dados e precisão relevantes, prestaria um grande avanço na pesquisa de buracos negros, facilitando a execução de simulações e dimanado o tempo de espera por dados.

O **grmonty** também apresenta algumas outras características que o tornam mais atra-

ente no quesito de programa que pode ser otimizado. Como ele é relativamente pequeno (menos de 5 mil linhas), possuir a característica de ter uma arquitetura SIMD e ser feito todo em linguagem C, uma portabilidade para ser executado em GPUs é um ato factível e que pode apresentar grandes ganhos com um esforço não muito alto, aproveitando assim da paralelização massiva que as placas gráficas apresentam.

Este trabalho tem como objetivo apresentar melhorias a execução do código do **grmonty**, utilizando-se do processador de placas gráficas, as GPUs, para massivamente paralelizar e distribuir a carga de trabalho pelos múltiplos núcleos de processamento destas placas. Primeiramente é explicado o que é o **grmonty** e como funciona, depois os paradigmas ao qual sua execução apoia-se, caminhando para a explicação de GPUs e como contribuem para o aumento de performance, assim são apresentadas as otimizações executadas, chegando finalmente nos resultados alcançados e conclusões. Há ainda um capítulo de próximos passos demonstrando que ainda há muito espaço para mais melhorias e mais velocidade na execução.

Capítulo 2

Grmonty: Monte Carlo para Relatividade Geral

2.1 O que Faz

Dolence et al definem o **grmonty** como "software destinado a calcular o espectro de plasmas quentes e opticamente finos a par da completa relatividade geral utilizando um código de transporte radioativo baseado na técnica de Monte Carlo"(Dolence *et al.*, 2009, p.1, traduzido). Em outras palavras o programa estima o espectro de uma simulação de magnetoidrodinâmica Alfvén (1942) relativística utilizando o método de Monte Carlo.

Utilizando o método de Monte Carlo na geração dos dados, os fótons, e a partir de um dado modelo de plasma fornecido como entrada, o qual especifica velocidade, densidade, força do campo magnético e temperatura, o programa busca gerar o espectrograma. Para tanto um número próximo a N - fornecido na entrada - de fótons é gerado e para cada fóton sua trajetória é traçada. Nessa trajetória o fóton é espalhado e pode passar por diferentes interações, até finalmente ser mensurado.

Depois de algumas iterações um número próximo a N de fótons já foi gerado e rastreado, assim um relatório com o espectrograma é obtido e retornado pelo programa que finalmente termina.

2.2 Para que Faz

Foram desenvolvidas várias técnicas para calcular a transferência radioativa a partir de fontes como as descritas a baixo(Dolence *et al.*, 2009), porém poucas levam em conta a relatividade geral como um todo, principalmente no quesito de objetos, fontes, muito massivas ou com velocidades próxima a da luz, o **grmonty** vem para aprimorar esses cálculos.

Qualquer fonte astrofísica de radiação que seja relativística, ou seja, qualquer corpo ou fenômeno fonte de radiação eletromagnética, seja do rádio à raios gama e que é relativística: apresenta uma considerável distorção no espaço-tempo, seja por estar em velocidades próxi-

mas a da luz, seja por possuir uma enorme quantidade de massa e/ou energia. Exemplos de objetos são os buracos negros e estrelas de nêutrons, fenômenos são os *Gamma Ray Bursts* ou núcleos ativos de galáxias.

2.3 Como Faz

No momento de criação e rastreio dos fótons o programa faz o uso da biblioteca **OpenMP** para paralelizar o desenvolvimento dos fótons, graças a esta abordagem é viável o potencial uso de todos os núcleos disponíveis na CPU da máquina. A biblioteca é utilizada para que cada fóton seja produzido e espalhado de forma independente dos outros e funcionando em paralelo, além disso todas as instruções não dependem do fóton em si, elas são as mesmas instruções para todos os fótons. Desta forma podemos caracterizar o **grmonty** como tendo uma computação SIMD.

"Única Instrução Múltiplos Dados: Nesse tipo de computação podem haver múltiplos processadores, cada um operando sobre seu item de dados, mas estão todos executando a mesma instrução naquele item de dados"(Eijkhout *et al.*, 2016, p.84, traduzido). A arquitetura SIMD trabalha em ressonância com o **OpenMP** uma vez que torna a paralelização muito simples de ser aplicada: não há variáveis compartilhadas, não há condicionais ou desvios de fluxo que tornem cada execução diferente uma da outra, não há necessidade de sincronização ou *mutex*. Tornar o programa paralelizável é simples já que requer um uso mínimo do ferramental de programação concorrente.

Toda a vez que um fóton é criado logo em seguida sua rota é traçada, a relação entre criação e cálculo de trajetória é de 1 para 1. O que é evidente ao se observar as linhas 106 a 137 do *grmonty.cu*, aqui copiadas:

```
1     #pragma omp parallel private(ph)
2     {
3       while (1) {
4         /* get pseudo-quanta */
5     #pragma omp critical (MAKE_SPHOT)
6         {
7           if (!quit_flag)
8             make_super_photon(&ph, &quit_flag);
9         }
10        if (quit_flag)
11          break;
12
13        /* push them around */
14        track_super_photon(&ph);
15
16        /* step */
```

```
17      #pragma omp atomic
18          N_superph_made += 1;
19          /*mais codigo*/
20        }
21      }
```

Fica claro também - ao observar a linha 8 a 14 - que o processamento do rastreio é feito assim que possível, ao oposto de um processamento em lotes, ou seja, assim que o comandado *make_super_photon* é executado, gerando um novo fóton *ph*, o *track_super_photon* é chamado, não havendo algum buffer ou lote, um fóton produzido é um fóton consumido.

Tal processamento reduz muito os vestígios que um fóton pode criar durante sua existência. Uma vez que não se perde tempo deixando-o na memória, assim que é mensurado seu espaço na memória já é ocupado pelo próximo fóton a ser produzido, há um foco na economia de memória. O número de fótons na memória é o número de threads rodando simultaneamente.

Por fim se faz necessário notar que o programa é escrito na linguagem de programação C. O que faz muito sentido do ponto de vista de performance, uma vez que C é uma linguagem de baixo nível, mais próxima a linguagem de máquina e por isso é quase sempre explícito a quantidade e de que forma se está manipulando a memória. Outras vantagens são as possibilidades de usar tanto a biblioteca **OpenMP** como as otimizações do **gcc**, o *Gnu C Compiler*, mas do ponto de vista da expressividade uma linguagem de mais alto nível poderia apresentar outras vantagens, como uma maior legibilidade do código e o uso de abstrações e encapsulamento, aumentando também a capacidade e a facilidade de fazer manutenções e melhorias no código. e

Capítulo 3

GPGPU

A seguir é explanado o que vem a ser uma placa gráfica, principalmente seu processador, a GPU. Depois é descrita a motivação para a qual esse hardware foi criado, o por quê ainda é produzido e por que provavelmente continuará. Seguindo são apontadas as diferenças arquiteturais entre uma CPU e uma GPU, vantagens e desvatagens. Na continuação são apresentadas e demonstradas as duas principais linguagens que permitem o acesso a programação genérica na GPGPU a OpenCL e a CUDA, contendo o por quê neste projeto foi escolhido a linguagem CUDA. Por fim são apresentadas aplicações que se utilizam do estado da arte no hardware e software das placas gráficas mais atuais e avançadas.

3.1 Placas de vídeo

Placa de vídeo é uma peça que pode ou não estar presente em um computador, ela é responsável por fornecer o hardware especializado em renderização gráfica, projetado para prover um grande aumento de performance a um baixo custo se comparado a CPUs no que disrespeito a tranformação de dados na memoria em imagens e vídeos, prontos para serem apresentados em telas e monitores. São constituídas principalmente de:

- Um chip de memória, cuja a principal função é armazenar texturas, ou qualquer outro dado que deverá ser várias vezes utilizado, consultado, no processo de renderização.
- Portas de acesso e conexão a monitores e telas, variam entre VGA (Video Graphics Array), DVI (Digital Visual Interface) e HDMI (High-Definition Muiltimidia Interface). São as saídas mais comuns às computações das placas.
- Um processador, uma GPGPU composta de alguns milhares de núcleos
- Uma ou várias ventoinhas, para dissipar o calor produzido pelo processador

[1]Disponível em https://commons.wikimedia.org/wiki/File:NVIDIA-GTX-1070-FoundersEdition-FL.jpg Domínio público

Figura 3.1: *Placa grafica NVIDIA GTX 1070 para computadores de mesa "desktop", essa versão é a "Founders edition", possui cerca de 8 GBs de memória com uma banda de 256 GB/s , 1920 núcleos, frequência de funcionamento de 1506 MHz e consome 150W. (Fonte: Wikimedia Commons [1])*

A figura 3.1 é uma foto de uma placa gráfica da Nvidia, a NVIDIA GTX 1070, começou a ser comercializada em Junho de 2016 à um preço de 379 dólares e possuí 1920 núcleos de processamento à 1,506 GHz em sua GPU (Balraj, 2016). Nessa mesma época também é comercializada o *Intel® Core™ i7-6850K Processor* CPU da Intel, vendida na época a partir de 617 dólares, possui 6 núcleos à uma frequência de 3,6 GHz (Intel, 2016). Um cálculo simples de tiques por segundo, ou seja, quantidade de ciclos que podem ser executados por segundo para cada processador demonstra que, enquanto a CPU da intel é capaz de realizar 21,6 bilhões de ciclos, no total, por segundo a GPGPU da nvidia é capaz de 2891,52 bilhões de ciclos, no total, por segundo, uma diferença de aproximadamente 133,8 vezes. A GPU que é 60,42% do valor da CPU é mais de 100 vezes mais rápida.

GPGPU é uma acrônimo para *General Purpose Graphics Processing Unit* em tradução literal: Unidade de processamento Gráfico de Propósito Geral. É uma evolução, uma adaptação que as GPUs passaram, na qual sua cadeia de processamento gráfico foi flexibilizada tornado possível usá-la para propósitos mais gerais, indo muito além do escopo de produção de gráficos e imagens em três dimensões, porém suas origens e modo de operação advem da sua função original de processar uma cadeia de dados gráficos, e tal necessidade determinou qual seria o objetivo da arquitetura que tais processadores deveriam ter. Assim entender essa cadeia de processamento ou *pipeline* dos dados das placas gráficas é importante para entender o por quê são como são (Kirk e mei Hwu, 2016).

O objetivo das GPUs é gerar imagens e vídeos que representam visões de uma cena virtual. A visão desta cena é descrita pela posição de uma câmera virtual e é definida pela geometria, orientação e propriedades materiais da superfície dos objetos na cena representados, bem como das propiedades das fontes de luz. APIs gráficas como OpenGL (The Khronos Group Inc., 2018a), DirectX (Microsoft, 2018) ou Vulkan(The Khronos Group Inc., 2018b) representam este processo como um pipeline que executa uma cadeia de operações sobre um conjunto de vértices enviados pela CPU, sendo que cada vértice possui propriedades determinadas, tais como cor, posição e vetor normal (Fatahalian e Houston, 2008).

O vertex shader é uma parte integrante dessa cadeia e está no início do processamento.É um programa que executa um conjunto de operações sobre cada um dos vértíces de entrada, projetando cada vértice, baseado em sua posição relativa à camera virtual, em um amteparo 2D. Destes vértices, é montado um conjunto de triângulos, que representam os objetos no espaço 2D. Assim, quanto maior a quantidade de triângulos, melhor a qualidade com que o objeto será representado (Fatahalian e Houston, 2008). Visto que cada cena possui milhares de vértices e cada um deles pode ser tratado independentemente, os vértices podem ser processados de forma paralela e/ou distribuída (GREEN *et al.*, 2008).

O proxímo passo é o processo de rasterização, consistindo em determinar quais espaços da tela são cobertos por cada triângulo. Esse processo resulta na geração de fragmentos para cada espaço de tela coberto. Um fragmento é o que virá a ser um pixel, que contém todas as informações necessárias para gerar uma imagem final como profundidade, posição no frame buffer, etc. A partir da localização da câmera virtual, os fragmentos que são ocultos por outros fragmentos são descartados, só é necessário mostras os objetos que podem ser vistos (GREEN *et al.*, 2008).

Por fim há o pixel shader que opera sobre a saída gerada pelo processo de rasterização. O pixel shader é um programa que consiste em um conjunto de operações que são executadas sobre cada fragmento, antes que estes sejam plotados na tela. Utilizando as informações de cor dos vértices e, possivelmente, buscando dados adicionais na memória global em forma de texturas (é nesse momento que a memória principal da placa é necessária), cada fragmento é processado para obter-se a cor final do pixel. Assim como na etapa de processamento de vértices, os fragmentos são independentes e podem ser processados em paralelo. Esta etapa é a que tipicamente demanda maior processamento dentro da estrutura do pipeline gráfico (GREEN *et al.*, 2008).

Uma vez que esses programas podem ser aplicados em milhares de vértices e pixels independentemente, as GPUs evoluíram para ser um conjunto de multiprocessadores massivamente paralelos. Além disso, dependendo do balanceamento da carga de trabalho das aplicações, apesar dos pixels serem dependentes dos vértices, ambos podem ser executados paralelamente. Esta característica resultou no aumento da programabilidade dos multiprocessadores da GPU (Kirk e mei Hwu, 2016).

Essa cadeia de processamento é o que destacou as GPUs das CPUs convencionais. No desenvolvimento de processadores dedicados a processar essa cadeia na forma mais eficiente

possível as GPUs tomaram um caminho mais focado na paralelização, com mais núcleos, mais unidades logico-aritméticas em detrerimento de mais ciclos por segundo, maiores frequencias de trabalho, priorizando uma única memória grande e não muito rápida ao invés de varias camadas de memória cache muito rápida.

Esta arquiterura muito focada, ganhou a atenção de outras computações não necessariamente ligadas a renderização de imagens. Tal demanda eclodiu na flexibilização das GPUs para GPGPUs.

3.2 A História das GPU e GPGPU

As GPUs, a princípio, não foram criadas para cálculos numéricos ou simulações de particulas e sim para a rederização de imagens para jogos digitais, o alto custo na produção e aquisição de hardware mais potente, pricipalmente a memória, na década de 70 impulsionou a criação de processadores mais dedicados e com funções mais específicas para a rederização gráfica. Impulsionada pelo mercado de jogos de video game, e buscando evitar o alto valor de chips de memória, placas de sistemas de arcade apresentavam um conjunto de chips de vídeo para combinar os dados enquanto estes eram escaneados e direcionados ao monitor (Hague, 2013).

Em 1977 o Atari 2600 usou um *video shifter*, um tipo de circuíto integrado responsável por emitir o sinal de TV, chamado *Television Interface Adaptor* ou TIA. Foi customizado para ser capaz de gerar as imagens finais para a tela, os efeitos sonoros e ler os comandos vindos dos joysticks, teve como direcionamento em seu design a economia de memória RAM, muito cara na época (Hague, 2013) (Atari, 1983).

Em 1985 o Commodore Amiga apresentou um chip gráfico que vinha com um circuíto *blitter*, para maior velocidade na manipulação de memória, tranformação de bitmaps, desenho de linhas e funções de preenchimento de áreas. Também vinha com um co-pocessador, capaz de executar instruções únicas e manipular os registradores em sincronia com o canhão de desenho dos tubos de raios catódicos das televisões (Jessen, 2017).

Na decada de 90 a nintendo e a sony criaram seus consoles de vídeo game, o nintendo 64 e o playstation, ambos capazes de produzir gráficos em 3 dimensões a partir de polígonos. A distição entre os dois principais processadores desses vídeo games eram claras, a nintendo já havia detalhado que seu sistema vinha com dois processadores: um de proposito mais geral o *MIPS R4200* e o *Reality Coprocessor* desenhado para cálculos em 3d de alta performance, pre-processamento de aúdio e vídeo, mapeamento de textura e buferização de profundidade em tempo real (Nintendoi of America Inc., 1998) (McCall, 1996).

A nintendo não utilizou o termo CPU para descrever seu *Reality Coprocessor*, foi a Nvidia em 1999 que popularizou o termo. Ele já existia desde década de 80, porém somente ápos a capanha de marketing de sua placa de vídeo, a GeForce 256 com o logo: *"the world's first GPU"* (A primeira GPU do mundo) que o termo ficou mais comum (NVIDA, 1999).

Em 2002 foi introduzido a placa de vídeo ATI Radeon 9700 (também conhecida como R300)(**?**) nela tanto o pixel shader quanto o vertex shader poderiam implementar laços e longas contas aritiméticas de ponto flutuante. A GPU foi se tornando tão flexivel quanto as CPUs, porém em ordens de magnitude muito mais rápida em operações sobres listas, vetores.

Em 2007 a Nvidia lançou a CUDA, uma extensão da liguagem de programação C que poderia ser compilada e executada na GPU, oficializando o início da programação heterogênea nas placas de vídeo. CUDA foi o primeiro modelo de programação para GPU amplamente adotada, logo no ano seguinte surgui o OpenCL que se tornou amplamente suportado. OpenCL é um padrão aberto definido pelo *Khronos Group* o qual permite o desenvolvimento de código tanto para a CPU quanto a GPU com enfasi na portabilidade (**?**). Programas nessa especificação podem rodar em dispositivos Intel, AMD, Nvidia e ARM.

Já em 2014 a Nvida lança o GameWorks, um *middleware* que facilita e aumenta a performance da renderização de particulas como fumaça e fogo, faz simulações de fluidos e cálculos de *ray-tracing* ou rastreamento de raios (**?**), enquanto a AMD lançou o TressFX, biblioteca que permite a criação de pelos e cabelos que aparencia muito próxima a realidade (**?**).

O gráfico 3.2 demonstra a performance de algumas das principais placas de vídeo da Nvidia em um período de 6 anos, de 2010 a 2016. O gráfico demonstra a tendencia ainda muito forte de crescimento cada vez mais veloz da performance das GPUs ao longo do tempo, como a demanda para processamento e performance gráfica é virtualmente infinita e além disso a cada nova versão a diferença da atual para a anterior é cada vez maior, não há previsão para o esfriamento na produção e desenvolvimento de GPUs mais e mais poderosas.

3.3 CPU vs GPU

Ambos os processadores tem a tarefa primordial de executar intruções manipulando a memória e performando cálculos aritimeticos, porém suas necessidades particulares e os casos de uso que se encotraram redefiriam sua arquitetura, agora mesmo tendo a mesma origem suas peças de hardware são profundamente distintas, cada uma com suas vantagens e desvatagens sobre a outra.

Já em 2009, o poder computacional para operações em ponto flutuante de uma GPU era cerca de dez vezes maior ao de uma CPU, ver seção 3.1. Já a sua taxa de transferência era cerca de três vezes superior (Kirk e mei Hwu, 2016). Uma CPU é otimizada para desempenho de código sequencial, geralmente utilizam-se de uma lógica de controle mais sofisticada, fazendo com que a CPU seja projetada com mecanismos como previsão de desvios e execução fora de ordem (Out-of-order execution (PDF). cs.washington.edu. 2006. Retrieved 17 January 2014. "don't wait for previous instructions to execute if this instruction does not depend on them"). Desta forma, permite-se que um único fluxo de instruções seja executado

[2]Disponível em https://www.techspot.com/article/1191-nvidia-geforce-six-generations-tested/page6.html visitada em 31/01/2018

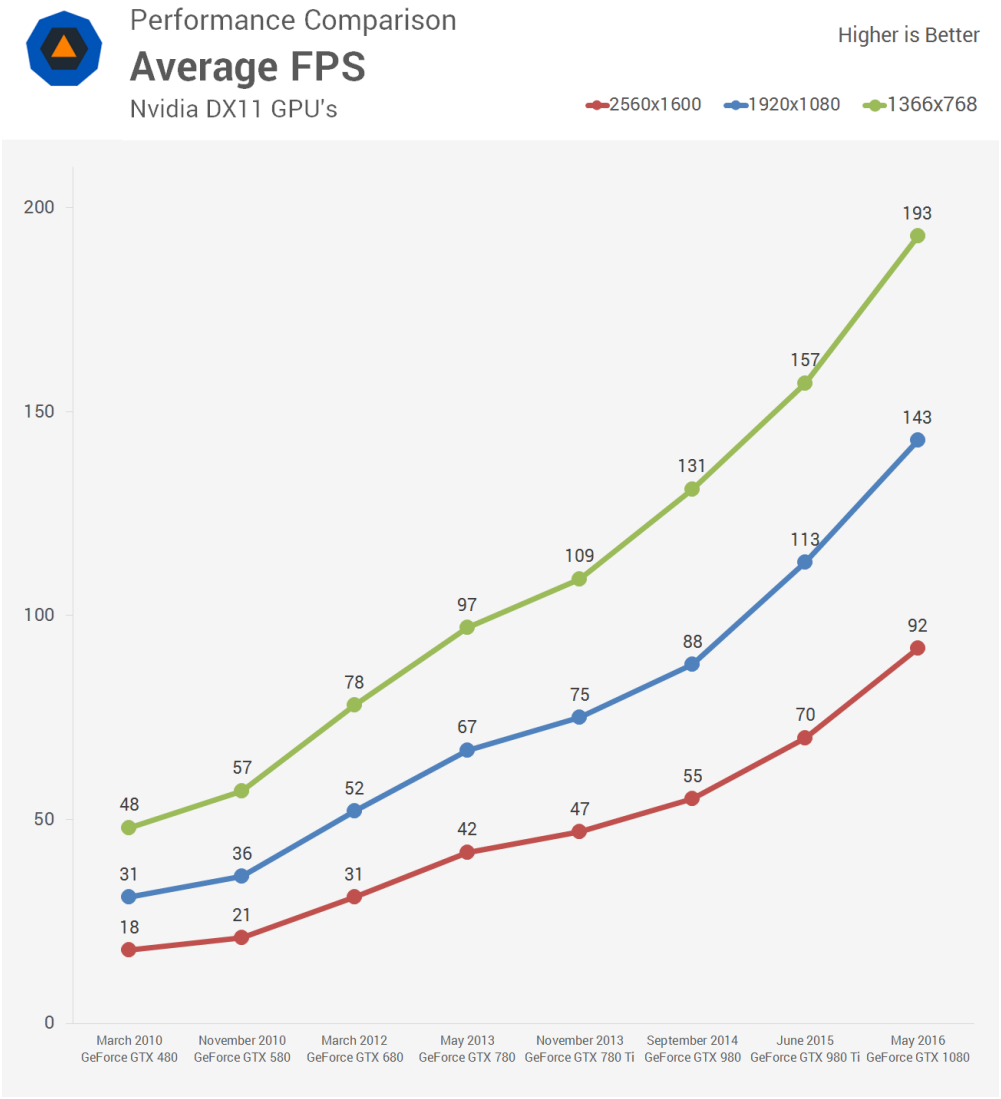

Figura 3.2: *média de fps (frames por segundo) em alguns jogo modernos para as principais placas de vídeo da Nvidia que foram lançadas de 2010 a 2016 . (Fonte: TECHSPOT [2])*

em paralelo enquanto a aparência de uma execução sequencial é mantida. Além disso, a CPU também é otimizada de forma a reduzir a latência de acesso aos dados e às instruções na memória principal. De fato, grande parte dos recursos da CPU são utilizadas para o gerenciamento de vários níveis de memória cache, que possuem o objetivo de minimizar o tempo necessário para que os dados requisitados sejam de fato utilizados pelo processador. Uma vez que a CPU tem como alvo programas de propósito geral, poucos de seus recursos são utilizados para processamento de operações de ponto flutuante (Kirk e mei Hwu, 2016).

Já a GPU foi concebida para ser a solução de problemas no qual processamento dos dados pode ser realizado massivamente em paralelo, ou seja, o mesmo programa é executado simultaneamente em muitos conjuntos de dados com alta intensidade aritmética (NVIDIA, 2010a). Por causa disto, as GPUs dedicam a maior parte de seus recursos ao processamento aritmético ao invés de cache de dados ou controle de fluxo (Kirk e mei Hwu, 2016).

Na Figura 3.3, é exposta de maneira simplificada com que os transistores do hardware

são distribuidos nos chips da CPU e GPU. Na imagem da esquerda há a representação de uma CPU, nela grande parte da área do chip é dedicada à memória cache e a lógica de controle. já na GPU, a imagem do lado direito, a maior parte dos recursos do chip é destinada ao processamento em paralelo de operações em ponto-flutuante. Assim, sua capacidade de processamento de dados é potencializada consideravelmente (NVIDIA, 2016).

Na GPU, grande parte dos dados é utilizada uma única vez, sendo que estes dados precisam ser transferidos da memória principal da GPU para o uso de seus multiprocessadores. Desta forma, o sistema de memória da GPU é projetado para maximizar a taxa de transferência de dados (throughput), ou seja, a GPU é projetada de forma a aumentar a largura de banda, ao invés de minimizar a latência para o acesso aos dados. Esta grande largura de banda é obtida através do uso de amplos barramentos e memórias dynamic random access memory (DRAM) do tipo graphics double data rate (GDDR) (Fatahalian e Houston, 2008). É importante destacar que nem todas as aplicações terão um desempenho melhor na GPU em relação à CPU. As GPUs são projetadas como engines de processamento numérico, e não executam bem os programas sequenciais que as CPUs normalmente executam. Assim, deve-se utilizar a placa GPU como um co-processador para auxiliar a CPU na execução dos programas, de forma que sejam atribuídas as partes sequenciais à CPU e as partes com processamento paralelo aritmético à GPU (Kirk e mei Hwu, 2016).

Figura 3.3: *Imagem comparando de forma simplifica a arquitetura dos processadores. Do lado esquerdo a arquitetura de uma Unidade de Processamento Central, muito espaço para o cache e a unidade de controle. Do lado direto uma unidade de processamento Gráfica, como uma grande região segmentada dedicada a computação aritimética e lógica. (Fonte: Wikimedia Commons [3])*

3.4 Bibliotecas: OpenCL e CUDA

Nos primórdios da computação genérica em GPUs duas formas principais surgiram para lidar com essa programação, a linguagem CUDA e a OpenCL.

Open Computing Language (OpenCL) é um arcabouço, um *framework*, aberto para programação genérica para vários processadores, não só GPGPUS como também CPUs e outros

[3]Disponível em https://commons.wikimedia.org/wiki/File:Cpu-gpu.svg Creative Commons Attribution 3.0 Unported

dispositivos focados em processamento. O OpenCL da suporte para sistemas embarcados, sistemas pessoais, corporativos e até de Computação de Alta Performance. Isso é alcançado graças a criação uma interface de baixo nível, ou seja, o mais próximo do hardware possível, mantendo um auto desempenho, com uma abstração portátil. O OpencL também é uma API para controle de aplicações paralelas em sistemas com processadores heterogêneos. O OpenCL consegue, numa mesma aplicação, reconhecer vários processadores e dispositivos diferentes dentro de um mesmo computador, ou cluster de computadores, e executar códigos distintos entre eles, coordenando os dispositivos. Desta forma, igual ao CUDA, a parte do código que é executada sempre na CPU, de forma sequencial é chamada de código *Host*. A parte que roda o *kernel*, o código que geralmente roda na GPGPU, preparado para rodar massivamente em paralelo é chamado de *device* . É importante lembrar que dado essa generalização do OpenCL, é possível que a CPU onde o código do host esteja executando seja também usada para rodar um kernel, e essa CPU passa a executar código device ao mesmo tempo em que roda código host. Tanto o fato do OpenCL ser aberto quanto o fato dele não se restringir a um hardware especifico fazem dele a linguagem mais usada para GPGPU fora de GPUs NVIDIA (Evan's data, 2011).

Compute Unied Device Architecture (CUDA) pode ser descrita como uma linguagem de programação para GPGPUs criada pela NVIDIA em 2007 para seus processadores G80, fornecidos nas placas GeForce 8800 Ultra, GeForce 8800 GTX, GeForce 8800 GTS(G80). Ele adiciona suas diretrizes para as linguagens C, C++, FORTRAN e Java, permitindo que elas usem a GPGPU. Esse trabalho usa o CUDA junto com a linguagem C.

A versão 1.0 do CUDA foi disponibilizada em Fevereiro de 2007. Atualmente só existe um compilador para CUDA, o nvcc, e ele só da suporte para GPUs NVIDIA. Para uma função executar na GPGPU esta precisa ser invocada de um programa da CPU. Chamamos esse programa de Host e a GPGPU onde o kernel executará de Device. O CUDA implementa um conjunto virtual de instruções e memória, tornando os programas retroativos. O compilador primeiro compila o código em C para um intermediário, chamado de PTX, que depois será convertido em linguagem de máquina. Na conversão do PTX para linguagem de máquina o compilador verica quais instruções o device suporta e converte o código para usar as instruções corretas. Para obter o maior desempenho possível, é importante saber para qual versão o código final será compilado, pois na passagem do código de uma versão maior para uma menor não existe a garantia que o código seguira as mesmas instruções, o compilador pode mudar um conjunto de instruções para outro menos efíciente, ou em alguns casos, algumas instruções não existem em versões mais antigas.

Uma grande diferença entre essas duas linguagens é a quantida e qualidade da documentação e recursos para o seu aprendizado. As duas apresentam basicamente a mesma estrutura e modelo de desenvolvimento, tornando razoavelmente simples para uma programadora ou um programador migrar de uma linguagem para a outra, porém a grande diferença nos recursos de aprendizado e documentação entre as duas, torna consideravelmente mais fácil e rápido aprender e desenvolver primeiramente com a linguagem CUDA.

Este é o principal motivador para que se tenha escolhido a linguagem CUDA para aumentar a performance do grmonty. O nosso grupo de estudos que é focado em aprimorar o grmonty tem a necessidade de ter um conjuto de pessoas com certo domínio sobre a linguagem que será usada no desenvolvimento, e graças a facilidade e agilidade do aprendizado da linguagem CUDA esta foi a escolhida que ser primeiramente empregada no aprimoramento de performance do grmonty.

3.5 Aplicações e usos avançados

No ano de 2017 não esta se observando uma retração no desenvolvimento das GPUs e sim uma expanção. Seja provocada pelo já conhecido mercado de jogos digitais, seja por novas fronteiras como mineração de criptomoedas ou novas técnicas de aprendizado de máquina como *deep learnig*. Abaixo são citadas algumas das aplicações e modelos de problemas que expandem o escopo e avançam o potêncial que as GPUs podem alcançar.

O aprendizado profundo, *Deep Learning* é uma área de grande crescimento em inteligência artificial, ajudando os desenvolvedores a terem um controle mais fino e compreender melhor dados, como imagens, som e texto. Usando redes neurais, esses sistemas têm a capacidade de ver, aprender e reagir a situações complexas tão bem quanto ou melhor que os humanos treinados em tais ações. Isso está levando a uma forma totalmente diferente de pensar sobre coleta dados, tecnologias, produtos e serviços que podem ser oferecidos.

As soluções de aprendizado profundo contam quase exclusivamente com a computação acelerada pelas GPGPUs para treinar e acelerar aplicações, como identificação de imagens, caligrafia e vozes. As plcas gráficas se destacam em pipelines paralelos e aceleram as redes neurais em até 10 a 20 vezes, reduzindo cada uma das muitas iterações de treinamento de dados de semanas para dias. As placas de vídeo têm acelerado o treinamento de redes neurais profundas em 50 vezes em apenas três anos — mais rápido do que a lei de Moore — com outras 10 vezes previstas para os próximos anos. A inovação nessa área está acontecendo a um ritmo acelerado, abrindo oportunidades em aplicações como robôtica, medicina e carros autodirigíveis.

A realidade virtual (VR) é uma forma masi profunda de simular cenários mais realistas. Normalmente usando-se de capacetes ou óculos específicos, as vezes em combinação com objetos físicos reais no ambiente, imagens são geradas para cada um dos olhos e são atualizadas conforme o usuário move sua cabeça, proporcionando a sensação de estar corporalmente presente na cena gerada por computador. A pessoa portando esse equipamento pode "olhar ao redor"e interagir com itens e propriedades do mundo virtual.

O processamento necessário para gerar estas cenas de realidade virtual é atualmente muito alto, normalmente sendo exclusivos de dispositivos de primeira linha, uma vez que para que ocorra uma experiencia razoavelmente imersiva se faz necessário haver um atraso desprezível na captação e apresentação das imagens, de acordo com o movimento do usuário,

e as imegens devem ser atualizadas a uma frequência altissima para que não ocorra enjoes da parte de qyuem usa tais aparelhos.

O processo de mineração de criptomoedas pode ser extremamente lucrativo se for feito de maneira muito rápida, uma vez que quanto mais blocos da *blockchain* vc processar maior a sua chance de ganhar uma moeda no processo. Com o crescimento do interesse em bitcoins minerados - pessoas que buscam processar a blockchain, realizando transações das moedas - concluiram que as GPUs das placas gráficas usadas primordialmente para jogos poderiam ser usadas para resolver de forma muito mais rápidas os hash necessários para processar a blockchain.

A busca e o empenho dos mineradoes em performar da melhor forma possível, empurrou o desenvolvimento das GPGPUs à criação de novas arquiteturas de placas gráficas e até a novos tipos de GPUs, quase ou nada focadas em gráficos. Esses novos dispositivos focados em mineração de criptomoedas se tornaram um novo foco de desenvolvimento e inovação na área de processadores massivamente paralelos de computação genérica, ou mais especificamente na resolução de hashes ou fatoração de números primos.

Capítulo 4

Otimizacao

4.1 Arquitetura

mostrar gráfico de processamento do grmonty apontar o track super photon como candidato a ser produzido no kernel

4.2 Melhorias e Modificações

4.2.1 Somente uma dimensão

matrix pra vetor

4.2.2 OpenMP e Concorrência

desligar o openmp

4.2.3 math.h

unix math pra nvida math

4.2.4 divisão e trabalho e parelelização

calculo de diviasão de trabalho na GPU

4.2.5 Processar em Lotes

de “assim que possível” “para processamento em lotes”

Capítulo 5

Resultados

5.1 Métricas e Medição

the old

5.2 Comparações

demonstrar o aumento de 100X na velocidade

Capítulo 6

Futuro

6.1 Outras Linguagens de Programação

rust, nim, python

6.2 Single Precision

Usar float ao invés de double

6.3 Novos Disposítivos e Particulariedades dos frabicantes: AMD e NVIDIA

cálculo discreto, artiqueturas diferentes

6.4 Arcabouços: Tensorflow

tensorflow TPU TensorProcessingUnit

6.5 Application-specific integrated circuit chips (ASICs)

O que são? Onde vivem? O que comem?

Capítulo 7

Conclusões

Calculos são importantes e o avanço da ciência depende de artiteturas de alta performance, gpus tem se apresentado competentes na realização de tais tarefas, e sua popular adoção facilita um maior acesso computação astrofísica, aumentando assim a velocidade do progresso cientifico.

Referências Bibliográficas

Alfvén(1942) H. Alfvén. Existence of Electromagnetic-Hydrodynamic Waves. *nature*, 150: 405–406. doi: 10.1038/150405d0. Citado na pág. 3

Atari(1983) Atari. *Field Service Manual: Model 2600/2600A Domestic (M/N)*, 1983. Citado na pág. 10

Balraj(2016) Tarun Balraj. Nvidia geforce gtx 1070 goes on sale. https://www.techpowerup.com/223293/nvidia-geforce-gtx-1070-goes-on-sale, jun 2016. último acesso em 31/01/2018. Citado na pág. 8

Dolence *et al.***(2009)** Joshua C. Dolence, Charles F. Gammie, Monika Mościbrodzka e Po Kin Leung. grmonty: A monte carlo code for relativistic radiative transport. *The Astrophysical Journal Supplement*, 184:387–397. Citado na pág. 1, 3

Eijkhout *et al.***(2016)** Victor Eijkhout, Edmond Chow e V. Robert Geijn. *Introduction to High Performance Scientific Computing*. Saylor Academy. Citado na pág. 4

Evan's data(2011) Evan's data. Apac development study. Relatório técnico, Evan's data Corporation. Citado na pág. 14

Fatahalian e Houston(2008) K. Fatahalian e M. Houston. Gpus: A closer look. Citado na pág. 9, 13

GREEN *et al.***(2008)** S. GREEN, M. HOUSTON, D. LUEBKE, J. D. OWENS, J. C. PHILLIPS e J. E STONE. Gpu computing. *Proceedings of the IEEE*, 96. Citado na pág. 9

Hague(2013) James Hague. Why do dedicated game consoles exist? https://web.archive.org/web/20150504042057/http://prog21.dadgum.com/181.html, set 2013. último acesso em 4/5/2015. Citado na pág. 10

Intel(2016) Intel. Intel® core™ i7-6850k processor. https://ark.intel.com/products/94188/Intel-Core-i7-6850K-Processor-15M-Cache-up-to-3_80-GHz, jun 2016. último acesso em 31/01/2018. Citado na pág. 8

Jessen(2017) Steen Jessen. Big book of amiga hardware. http://www.bigbookofamigahardware.com/, 2017. último acesso em 31/01/2018. Citado na pág. 10

Kirk e mei Hwu(2016) David Kirk e Wen mei Hwu. *Programming Massively Parallel Processors*. Morgan Kaufmann, 3 edição. Citado na pág. 8, 9, 11, 12, 13

McCall(1996) Scott McCall. N64's us launch. http://www.pennoaks.net/archive64/Miscellaneous_Articles/N64_US_Launch.htm, sep 1996. último acesso em 31/01/2018. Citado na pág. 10

Microsoft(2018) Microsoft. DirectX. https://msdn.microsoft.com/en-us/library/windows/desktop/hh309467, 2018. último acesso em 31/01/2018. Citado na pág. 9

Nintendoi of America Inc.(1998) Nintendoi of America Inc. *Nintendo64 Function Reference Manual*, 1998. Citado na pág. 10

NVIDA(1999) NVIDA. Nvidia launches the world's first graphics processing unit: Geforce 256. https://www.nvidia.com/object/IO_20020111_5424.html, 1999. último acesso em 31/01/2018. Citado na pág. 10

NVIDIA(2016) NVIDIA. Nvidia tesla p100: The most advanced datacenter accelerator ever built featuring pascal gp100, the world's fastest gpu. Relatório Técnico WP-08019-001, NVIDIA. URL https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf. Citado na pág. 13

The Khronos Group Inc.(2018a) The Khronos Group Inc. OpenGL. https://www.opengl.org/about/, 2018a. último acesso em 31/01/2018. Citado na pág. 9

The Khronos Group Inc.(2018b) The Khronos Group Inc. Vulkan. https://www.khronos.org/vulkan/, 2018b. último acesso em 31/01/2018. Citado na pág. 9