

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

Caio Costa Salgado

# **Uso de GPGPU na Análise de Buracos Negros**

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

<b>Lista de Abreviaturas</b>	<b>vii</b>
<b>1 Introdução</b>	<b>1</b>
<b>2 Grmonty: Monte Carlo para Relatividade Geral</b>	<b>3</b>
2.1 O que Faz . . . . .	3
2.2 Para que Faz . . . . .	3
2.3 Como Faz . . . . .	4
<b>3 GPGPU</b>	<b>7</b>
3.1 História das GPU e GPGPU . . . . .	7
3.2 Bibliotecas: OpenCL e CUDA . . . . .	7
<b>4 Otimizacao</b>	<b>9</b>
4.1 Arquitetura . . . . .	9
4.2 Melhorias e Modificações . . . . .	9
4.2.1 Somente uma dimensão . . . . .	9
4.2.2 OpenMP e Concorrência . . . . .	9
4.2.3 math.h . . . . .	9
4.2.4 divisão e trabalho e paralelização . . . . .	9
4.2.5 Processar em Lotes . . . . .	9
<b>5 Resultados</b>	<b>11</b>
5.1 Métricas e Medição . . . . .	11
5.2 Comparações . . . . .	11
<b>6 Futuro</b>	<b>13</b>
6.1 Outras Linguagens de Programação . . . . .	13
6.2 Single Precision . . . . .	13
6.3 Novos Dispositivos e Particularidades dos fabricantes: AMD e NVIDIA . . . . .	13
6.4 Arcabouços: Tensorflow . . . . .	13
6.5 Application-specific integrated circuit chips (ASICs) . . . . .	13

<b>7 Conclusões</b>	<b>15</b>
<b>Referências Bibliográficas</b>	<b>17</b>



# Lista de Abreviaturas

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



# 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).

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 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, 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.

Este trabalho tem como objetivo apresentar melhorias a execução do código do **grmonty**, utilizando-se do processador de placas gráficas, as GPGPUs, 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 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 programa busca gerar o espectro de radiação. 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 passa por diferentes interações, podendo ser absorvido, refletido, difratado ou até re-emitido ao percorrer seu percurso e 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 neutrons, 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”(Victor *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 rastreo é feito assim que possível, ao oposto de um processamento em lotes, ou seja, assim que o comando *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.





# Capítulo 3

## GPGPU

### 3.1 História das GPU e GPGPU

era uma vez...

### 3.2 Bibliotecas: OpenCL e CUDA

o que são e como funcionam porque escolhemos cuda?



# 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 paralelização

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 Dispositivos e Particularidades dos fabricantes: AMD e NVIDIA

cálculo discreto, arquiteturas 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 científico.



# Referências Bibliográficas

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