# EP2: Criptografia em GPUs usando CUDA

Gabriel Baptista 8941300

Hélio Assakura 8941064

Junho de 2017

## 1 Introdução

O conteúdo deste relatório consiste na análise do impacto da utilização da ferramenta `CUDA` para a execução de algoritmos[1] simples de encriptação e na resposta das perguntas feitas no enunciado do EP[2]. Os resultados serão mostrados em forma de gráfico de barras gerados usando a biblioteca `matplotlib` da linguagem `Python`.

Para a implementação, foi necessário instalar o `CUDA Toolkit`[3]. Os algoritmos escolhidos para serem executados em GPU foram: *ROT-13*, *Cifra XOR*[4] e *base64*. A implementação da `Cifra XOR` não se encontra no repositório, mas pode ser facilmente implementada.

## 2 Especificações

Para cada algoritmo, foram realizadas 30 medições. Os testes com `CUDA` foram realizados na placa de vídeos `NVIDIA GeForce GTX 560 Ti`, e os testes dos algoritmos em `C` executados com processador `Intel Core™ i7-2600K CPU @ 3.40GHz×8`, com 8GB de memória RAM.

A GPU da placa de vídeo possui as seguintes especificações mostradas na 1. As informações completas da placa podem ser encontradas em `http://www.nvidia.com.br/object/product-geforce-gtx-560ti-br.html`[Acessado em 11/06/2017].

Os arquivos usados para o teste de desempenho eram cópias do arquivo *king_james_bible.txt*[5], com seu conteúdo multiplicado diversas vezes. A quantidade de linhas de cada arquivo eram (50, 100, 200, 500mil, 1, 2, 5, 10 milhões).

[1] `https://github.com/phrb/MAC5742-0219-EP2/tree/master/src/crypto-algorithms`[Acessado em 11/06/2017]

[2] `https://github.com/phrb/MAC5742-0219-EP2/blob/master/doc/enunciado_ep2.pdf`[Acessado em 11/06/2017]

[3] `https://developer.nvidia.com/cuda-toolkit`[Acessado em 11/06/2017]

[4] `https://en.wikipedia.org/wiki/XOR_cipher`[Acessado em 11/06/2017]

[5] `https://github.com/phrb/MAC5742-0219-EP2/blob/master/src/crypto-algorithms/sample_files/king_james_bible.txt`[Acessado em 11/06/2017]

A versão em `C` não foi paralelizada, e tanto para essa quanto para a implementação em CUDA não foi usada nenhuma flag de otimização.

Devido a importância da verificação de erros na execução de algoritmos na GPU, as mensagens de verificação de erro e de informações da execução foram mantidas na realização dos testes.

| NVIDIA GeForce GTX 560 Ti | |
|---|---|
| CUDA Cores | 384 |
| Graphics Clock (MHz) | 822 |
| Processor Clock (MHz) | 1645 |
| Texture Fill Rate (billion/sec) | 52.5 |

Tabela 1: Especificações da GPU

# 3 Algoritmos

## 3.1 ROT-13

O código em CUDA desenvolvido foi uma adaptação do código base disponibilizado pelo monitor[6] para executar na GPU. Ele contém diversas mensagens e verificações de erros, o que é de extrema importância e foi mantido no código final. O código `C` foi modificado também para receber um arquivo de texto.

## 3.2 Cifra XOR

Como descrito em `https://en.wikipedia.org/wiki/XOR_cipher`:

> "In cryptography, the simple XOR cipher is a type of additive cipher, an encryption algorithm that operates according to the principles:
>
> $$A \oplus 0 = A \tag{1}$$
> $$A \oplus A = 0 \tag{2}$$
> $$(A \oplus B) \oplus C = A \oplus (B \oplus C) \tag{3}$$
> $$(B \oplus A) \oplus A = B \oplus 0 = B \tag{4}$$
>
> where $\oplus$ denotes the exclusive disjunction (XOR) operation."

[6] `https://github.com/phrb/MAC5742-0219-EP2/blob/master/src/crypto-algorithms/rot-13.c` [Acessado em 11/06/2017]

Baseado nisso, foi desenvolvido o algoritmo que encripta um arquivo fazendo a operação XOR (êm `C`) entre o arquivo e uma `Key`, escolhida pelo dono do arquivo, e para decriptar, realizar essa operação novamente com a mesma `Key`. Assim como no algoritmo *ROT-13*, as mensagens no código CUDA foram mantidas.

Para os testes, a `Key` usada foi o livro *Ulysses*[7].

# 4 Resultados

Os tempos obtidos na execução do programa sequencial e nos programas em CUDA foram bem diferentes. Quando o tamanho do texto era pequeno (50 mil linhas, por exemplo), o algoritmo sequencial teve um desempenho melhor (figuras 3 e 4), pois a cópia do vetor do `host` para o `device` é custosa. Porém, nas entradas maiores, podemos ver claramente o ganho de rapidez usando a GPU (figuras 1, 2).

O desvio padrão das medições sem output foi bem baixo. Grande parte dos testes ficou abaixo de 1%. Apenas medições com arquivos pequenos mostrou leve aumento no desvio, chegando a 3%. Já com arquivo de saída, aumentou para cerca de 5% para a maioria dos testes com *ROT-13* e não houve mudança significativa com a *Cifra XOR*.

# 5 Respostas às perguntas

As perguntas a serem respondidas são:

**1**) Como o tempo de execução varia conforme o tamanho do arquivo?

**2**) Como e por quais razões vocês escolheram o tamanho de block e grid ?

**3**) Qual o impacto das operações de I/O e alocação de memória no tempo de execução?

1. Ao realizar os testes, vimos que o desempenho do algoritmo em CUDA é bem melhor. Enquanto a média de tempo para a execução do algoritmo ROT-13 para o maior texto é de aproximadamente 11 segundos, em CUDA, é de 0.7 e na Cifra XOR, demora 4 segundos para o sequencial e 1 para o CUDA. Essa diferença se dá pois no ROT-13, são realizadas várias comparações e atribuições, e na Cifra XOR, apenas 1 operação é realizada. Ou seja, se o algoritmo realizar mais operações, a GPU começa a ficar mais atraente. Vale notar que ambos os algoritmos realizaram a cópia do texto do `host` para o `device` apenas uma vez.

[7] `https://github.com/phrb/MAC5742-0219-EP2/blob/master/src/crypto-algorithms/sample_files/ulysses.txt`

2. O tamanho do *Block* foi determinado automaticamente utilizando o comando `cudaOccupancyMaxPotentialBlockSize`()[8]. O tamanho do *Grid* foi decidido levando em conta o tamanho do texto e o tamanho do bloco, da forma

$$gridSize = \frac{numElements + blockSize - 1}{blockSize} \tag{5}$$

Em que *gridSize* é o tamanho do `Grid`, `numElements` é o número de caracteres no texto e `blockSize` o tamanho do *Block*. O tamanho máximo do grid é $2^{16} - 1$.

3. A operação de cópia do `host` para o `device` é muito custosa (figuras 3 e 4). Caso o algoritmo precise realizar essa cópia poucas vezes, o desempenho será muito bom, mas se for necessário fazê-la diversas vezes, a GPU pode não ser tão boa. Também podemos ver que operações de saída são bastante custosas, dependendo do algoritmo. No *ROT-13*, houve um aumento de até 5 segundos (para texto de 10 milhões de linhas), pois o arquivo de saída encriptado é bem grande (cerca de 433MB). Já na *Cifra XOR*, o arquivo gerado depende da `Key`, mas o tamanho é pequeno e não tem impacto tão grande no tempo final.

# 6 Conclusão

Ao desenvolver os algoritmos de encriptação em CUDA e compará-los com sua implementação em `C`, vimos que o desempenho da versão rodada na GPU é mais rápida, mas deve-se levar em conta diversos outros fatores, como a quantidade de operações de cópia que o algoritmo CUDA deverá realizar, a complexidade do desenvolvimento para a plataforma e a quantidade de dados que serão processados.

Foram implementados algoritmos de encriptação, mas GPUs também podem ser usadas para a decriptação. Vários algoritmos de decriptação são executados usando GPUs e também são extremamente mais eficientes que quando executados em CPUs.

Podemos concluir que, dependendo do algoritmo e da sua implementação, o uso de GPUs acelera consideravelmente o processamento de dados.

[8] `http://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__HIGHLEVEL.html#group__CUDART__HIGHLEVEL_1gee5334618ed4bb0871e4559a77643fc1`[Acessado em 11/06/2017]

# 7 Gráficos

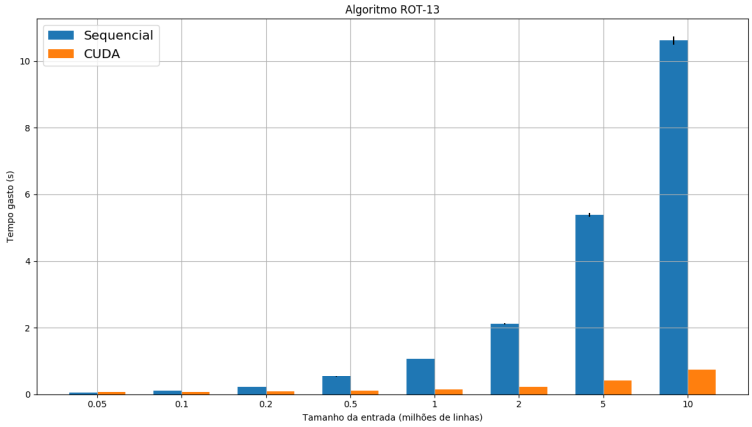


Figura 1: Tempo de execução do algoritmo ROT-13

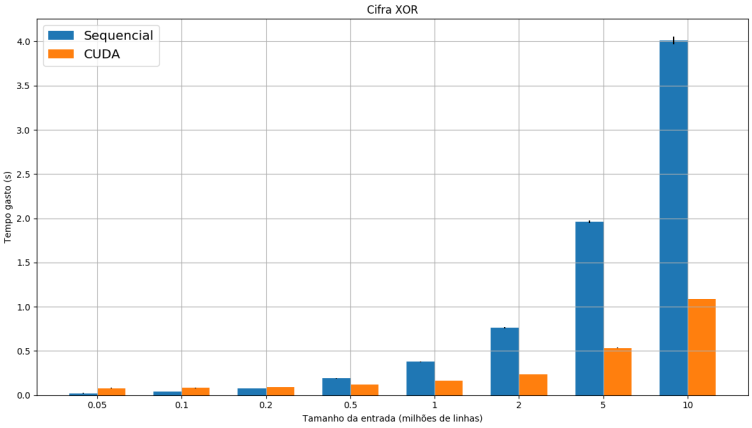


Figura 2: Tempo de execução da Cifra XOR

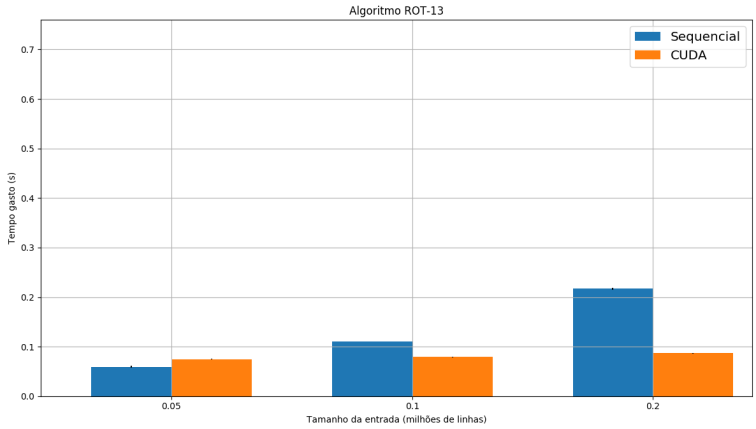


Figura 3: Tempo da execução do algoritmo ROT-13 para arquivos menores

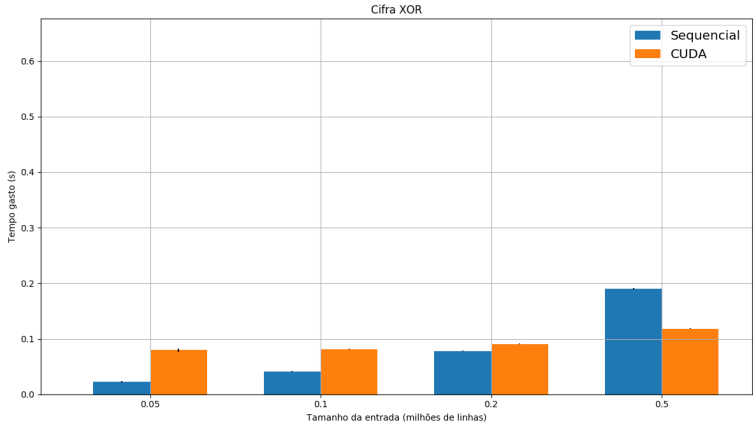


Figura 4: Tempo da execução da Cifra XOR para arquivos menores