Disciplinarum Scientia. Série: Naturais e Tecnológicas, Santa Maria, v. 16, n. 1, p. 83-94, 2015.
Recebido em: 11.04.2015. Aprovado em: 30.06.2015.

ISSN 2176-462X

PARALELIZAÇÃO DO ALGORITMO AES E ANÁLISE SOBRE GPGPU[1]

PARALLELIZATION OF AES ALGORITHM AND GPU ANALYSIS

Douglas Tybusch[2] e Gustavo Stangherlin Cantarelli[3]

RESUMO

Este trabalho trata da implementação e da análise do algoritmo criptográfico AES de forma paralelizada pelo uso de GPUs, utilizando a tecnologia CUDA para a obtenção de maior velocidade nos processos de criptografia. Para isso, foi criado um comparativo de performance de processamento do algoritmo entre uma CPU e uma GPU. Os resultados mostram um ganho favorável em tempo de processamento para implementações paralelizadas em GPUs, apesar da existência de grandes gargalos na transferência de dados para esses dispositivos. Este comparativo se mostra necessário devido ao grande avanço de processamento geral das últimas gerações de GPU, como também devido ao aumento da utilização de criptografia para a transferência e armazenamento de dados.

Palavras-chave: computação paralela, criptografia, CUDA, performance.

ABSTRACT

This paper deals with the implementation and analysis of the AES cryptographic algorithm in a parallelized way with the usage of GPUs and the CUDA technology in order to achieve greater speed in the encryption process. For this purpose, it was created a comparison for the processing performance of the algorithm between a CPU and a GPU. The results show a favorable gain in processing time for the parallelized implementation in GPUs, despite the existence of major bottlenecks for data transfer to such devices. This comparison proves necessary due to the advancement of general processing in the latest GPU generations, as it is also due to the increased use of encryption for data transfer and storage.

Keywords: *parallel computing, cryptography, CUDA, performance.*

[1] Trabalho Final de Graduação - TFG.

[2] Acadêmico do Curso de Sistemas de Informação - Centro Universitário Franciscano.

[3] Orientador - Centro Universitário Franciscano.

INTRODUÇÃO

No passado, as ferramentas utilizadas para desenvolver *softwares* para *GPUs* (*Graphical Processing Unit*) eram muito específicas, limitando-se a renderização de gráficos. Em 2006, a desenvolvedora de *chips* NVIDIA lançou, com sua nova geração de placas de vídeo, o CUDA (*Compute Unified Architecture*), uma arquitetura de computação unificada, como uma extensão para a linguagem de programação C, a qual permitia a "conversão" de programas *single-thread* em programas paralelos a serem executadas em suas GPUs (LUKEN; OUYANG; DESOKY, 2009).

Com a crescente demanda de transferências de dados, surge a necessidade de armazenar dados de forma segura e acessá-los de forma rápida. Embora o método de criptografia AES *(Advanced Encryption Standard)* seja seguro e eficiente, a quantidade de dados a serem criptografados simultaneamente é grande e crescente, gerando tempos de resposta cada vez mais baixos (LUKEN; OUYANG; DESOKY, 2009).

Este trabalho buscou desenvolver um *software* para a criptografia de dados de forma paralela, gerando um comparativo de performance entre o modo clássico de processamento via CPU e o método de processamento paralelo através da utilização de GPGPU (Computação de Propósito Geral na Unidade de Processamento Gráfico), pelo uso de tecnologias como o CUDA em conjunto com a linguagem de programação Visual C++ 11.0.

CUDA E ARQUITETURA DE PROGRAMAÇÃO GPGPU

CUDA é uma plataforma computacional e modelo de programação desenvolvido pela NVIDIA, com o objetivo de facilitar o acesso ao processamento paralelo via GPU, através de linguagens como C, C++ ou Fortran (NVIDIA, 2014a). O CUDA inclui uma linguagem de programação e sintaxe baseada no C, utilizando seu próprio compilador, o NVCC, baseado no compilador C/C++, capaz de executar a grande maioria de códigos C/C++, para o desenvolvimento dos *kernels* a serem executados na GPU (NVIDIA, 2014b).

Para satisfazer as necessidades de programação paralela e a utilização de *kernels* do CUDA, a linguagem foi estendida com diversos recursos, como funções para sincronização de *threads*, paralelização de operações e vetores e maior estruturação na alocação de dados na memória. Também foram criadas funções para determinação de características de um dispositivo, como capacidade de memória e quantidade de núcleos e *threads* de uma GPU (NVIDIA, 2014c).

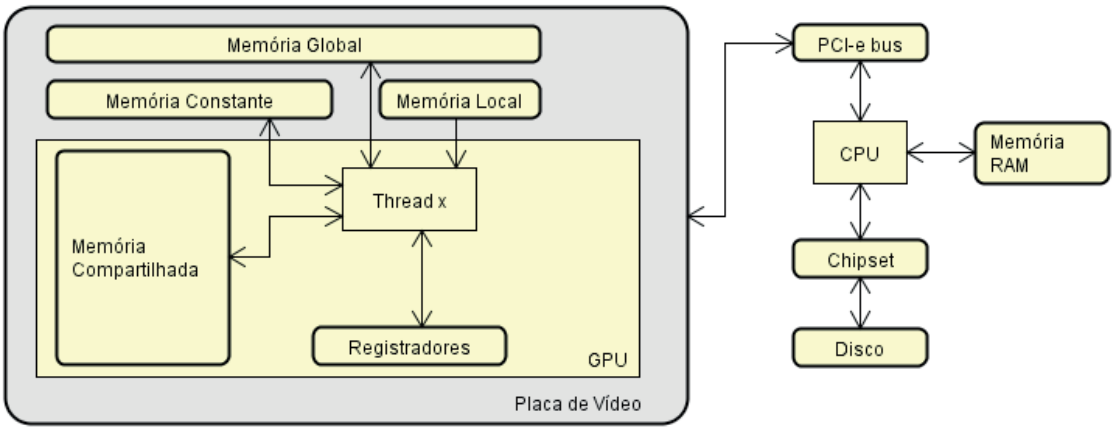
A estrutura de memória de uma GPU é representada no CUDA pelas seguintes regiões:

- Memória global: espaço de maior capacidade de um dispositivo, em que os dados são visíveis a todos os *threads/kernels* alocados, é também a região de acesso mais lento, devido à sincronização global de dados.

- Memória Constante: armazenada pelo mesmo banco de memórias da memória global, é utilizada apenas para leitura, sendo otimizada através de cache para uma maior velocidade de acesso.
- Memória Local: também armazenada pelo mesmo banco de memória global, é utilizada para escrita e leitura dentro do escopo de um *thread*.
- Registradores: armazenados no chip, são memórias de baixa quantidade e de acesso extremamente rápido. São de uso exclusivo de suas *threads* e armazena dados como endereços de variáveis e ponteiros.
- Memória Compartilhada: armazena dados a serem compartilhados entre *threads* de um mesmo bloco, a fim de reduzir o acesso a dados da memória global. Esta memória é armazenada no próprio chip gráfico.

Na organização de memórias, tanto as GPUs como as CPUs possuem seus próprios caches e registradores para acelerar o acesso a dados durante o processamento. No entanto, GPUs também possuem sua própria memória principal e endereçamento, significando que os dados a serem processados devem ser explicitamente transferidos para a memória principal da GPU, a partir da memória RAM de um computador, conforme exibido na figura 1.

Figura 1 - Fluxo de leitura de um arquivo do disco para a memória principal de uma CPU (arquitetura Intel Lynnfield), e para a memória global de um GPU (arquitetura Nvidia Kepler).



No CUDA, os *CUDA-cores* são a menor entidade de execução (equivalentes a *threads*), os quais pertencem a um bloco (um grupo *threads*) em que todos *threads* pertencentes a um mesmo bloco executam um *kernel* em comum (NVIDIA, 2014c). A vantagem da organização por blocos deve-se à habilidade de sincronia e compartilhamento de dados de forma eficiente entre os *threads* agrupados, pelo uso da memória compartilhada.

A diferença de *kernels* para blocos ou *CUDA-cores* pode ser dada como:

- *Kernels*: representam códigos e funções diferentes a serem executadas;

- Blocos: agrupamento de instâncias de um mesmo *kernel*, que executam um código comum, agrupados de forma a compartilhar recursos;
- *CUDA-cores*: *threads* de execução instanciados a partir de um *kernel*.

No CUDA, devido ao funcionamento de seu compilador, é possível que operações de memória sejam reordenadas para utilização de forma otimizada da arquitetura do dispositivo alvo. Assim, operações de acesso à memória podem não ocorrer de forma ordenada conforme o código fonte (NVIDIA, 2014c).

A sincronia de memória pelo CUDA pode ser feita pela utilização de diferentes funções que garantem que escritas e acessos na memória estejam em sincronia com ações feitas por outras *threads* em diferentes níveis de sincronização (por bloco de *threads*, todas *threads* de uma GPU, ou de GPUs e CPUs).

No CUDA, também são disponíveis funções para a sincronização de *threads*, garantindo que *threads* de um mesmo bloco alcancem um ponto específico de execução do *kernel*, antes de seguir o restante do código.

ADVANCED ENCRYPTION STANDARD

Advanced Encryption Standard (AES) é uma especificação para criptografia de dados estabelecido pelo *National Institute of Standards and Technology* (NIST, 2001). O AES foi baseado no algoritmo Rijndael, um algoritmo de bloco de cifras simétrico. Por simétrico, entende-se que se utiliza da mesma chave para ambos os processos, de criptografia e descriptografia de dados; e, por bloco de cifras, que se utiliza de blocos de dados de tamanho fixo de *bits* (NIST, 2001). A especificação AES possui algumas modificações em relação ao algoritmo original, como a fixação do tamanho do bloco de dados em 128 bits e chaves criptográficas de 128, 192 ou 256 *bits*, em que no algoritmo original poderia utilizar chaves e blocos de qualquer número múltiplo de 32 entre 128 e 256 *bits* (DAEMEN; RIJMEN, 2002).

No AES, os dados a serem tratados são processados como um bloco de dados de 128 *bits*, o equivalente a 16 *bytes*. O processo de criptografia se dá a partir da passagem do bloco de dados por diversas operações, as quais realizam sua conversão, descritas a seguir:

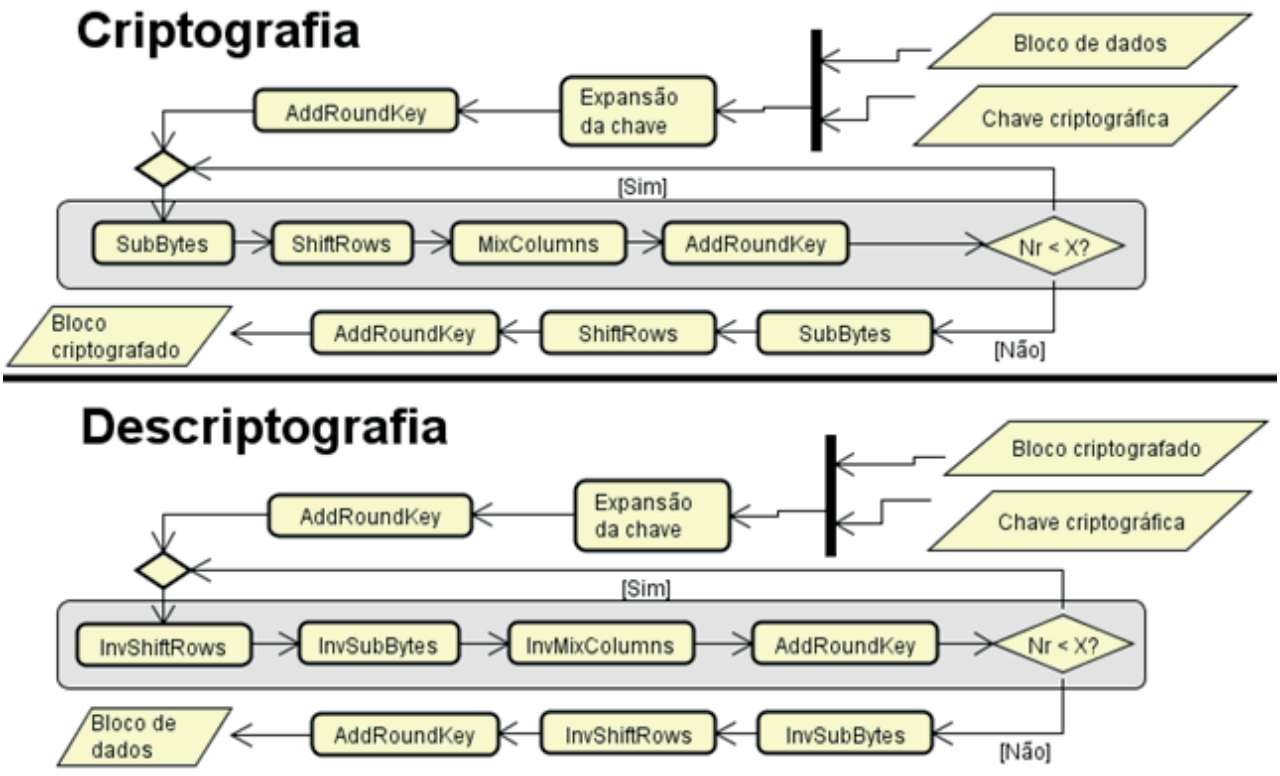
- Expansão da Chave: processo no qual são geradas diversas *round-keys* conforme o algoritmo de *Rijndael's key schedule*, em que a primeira *round-key* é derivada diretamente da chave criptográfica e cada *round-key* subsequente, baseada na *round-key* anterior;
- *AddRoundKey*: processo no qual cada *byte* do bloco de dados é combinado com o respectivo *byte* da *round-key* (demais *rounds*), por meio da operação X-OR;
- *ShiftRows*: processo no qual os *bytes* de cada linha são deslocados de forma cíclica N elementos à esquerda; com N começando em 0 (sem substituições) e aumentando em 1 a cada linha do bloco de dados;

- *Mixcolumns*: função na qual os *bytes* de uma coluna são combinados utilizando uma transformação linear invertível, multiplicando os dados desta coluna por uma matriz preestabelecida, seguindo os princípios de *Rijndael's Galois Field* e cuja implementação pode ser reduzida à substituição dos valores destes *bytes* por valores em tabelas preestabelecidas;
- *SubBytes*: é feita uma operação de substituição, em que cada byte é substituído por outro equivalente de uma tabela preestabelecida chamada *S-box*.

Para o processo de criptografia, primeiramente é feita a operação de expansão da chave criptográfica e, em seguida, a passagem do bloco pela operação de *AddRoundKey*. Após, o bloco é processado por um número fixo de iterações sobre as operações *SubBytes*, *ShiftRows*, *MixColumns* e *AddRoundKey*. Este número é baseado no tamanho da chave, sendo de 10, 12 e 14 iterações para chaves de 128, 192 e 256 *bits,* respectivamente. Ao final desse processo, é obtido, então, o bloco de dados em estado criptografado.

Para a descriptografia, é necessário o bloco já criptografado como entrada, que então passa por um processo semelhante, utilizando funções inversas equivalentes, conforme estipulado pela especificação AES, resultando, assim, no bloco de dados em seu formato original. Na figura 2, é possível visualizar seus respectivos processos.

Figura 2 - Fluxograma dos processos de criptografia e descriptografia do algoritmo AES.



IMPLEMENTAÇÃO

Este trabalho buscou desenvolver um aplicativo de execução via *prompt* de comando, criado a partir de Visual C++ 11.0, em conjunto da API CUDA 6.0. Possui como principais funções o carregamento de arquivos para a memória, a execução de criptografia e descriptografia dos arquivos, a opção para armazenamento dos arquivos criptografados e/ou descriptografados e a seleção dos modos de criptografia ou descriptografia dos arquivos:

- Via CPU: A CPU realiza todas as operações de criptografia e descriptografia e o carregamento de dados. O modo CPU *single-thread* representa um modelo de funcionamento no qual o arquivo é processado de modo sequencial em blocos de 16 *bytes*, enquanto o modo *multi-thread* permite a inicialização de N *threads*, em que cada *thread* realiza o processamento de diferentes blocos do arquivo, de modo paralelo.
- Via GPGPU: Neste modo são feitos o carregamento de arquivos pela CPU, a transferência dos arquivos para a GPU e a execução das operações de criptografia e descriptografia pela GPU, de forma semelhante ao modo CPU *multi-thread*.

O processo de paralelização do algoritmo AES no modo GPGPU constitui-se da criação de *kernels* CUDA, que executam as funções criptográficas do padrão AES, contendo pontos de sincronia, como a expansão da chave criptográfica, a alocação de tabelas em memória compartilhada e a transferência de arquivos na memória global da GPU, a partir da memória RAM do computador.

O algoritmo para criptografia de dados foi baseado em um pacote de códigos fonte criado e implementado por Vincent Rijmen, Antoon Bosselaers e Paulo Barreto, que serviu como a implementação base do padrão AES pelo NIST (ERDELSKY, 2002).

MODO CPU *SINGLE-THREAD* E *MULTI-THREAD*

Inicialmente implementado para a criptografia ou descriptografia de *strings* em blocos fixados de 16 *bytes,* foram necessárias algumas modificações nas funções do algoritmo original, para o processamento de arquivos e sua paralelização.

Para o processamento de arquivos, foram desenvolvidas funções para o carregamento de arquivos locais, em que estes são alocados dinamicamente e armazenados na memória RAM do computador, para uso das funções de processamento de dados. Como o algoritmo realiza a criptografia em blocos de 16 *bytes*, o tamanho do arquivo alocado é arredondado para o próximo valor múltiplo de 16 durante sua alocação.

As funções originais para criptografia e descriptografia foram modificadas de forma a receber um ponteiro do arquivo alocado, assim como um índice indicando o bloco do arquivo a ser processado.

Para o modo *single-thread* via CPU, após o carregamento do arquivo, é feita a operação de criptografia ou descriptografia do arquivo, por meio um laço de repetição que percorre a variável, processando-a em blocos de 16 em 16 *bytes*.

Para a paralelização do algoritmo AES em modo CPU *multi-thread*, foram desenvolvidas funções para a inicialização de N *threads* paralelos, nas quais o valor de N é baseado no número de núcleos da CPU. Após definido o número de *threads* a ser utilizado, são criados *structs*, os quais armazenam o endereço do arquivo a ser utilizado, como também seus índices de início e fim, representando seções do arquivo a serem processados pela *thread*.

Os valores de início e fim são encontrados a partir da divisão do tamanho do arquivo em *bytes*, pela quantidade de *threads* a serem inicializadas, também de modo a gerar seções de tamanhos múltiplos de 16.

Após a definição destas *structs*, é feita a inicialização das *threads* para processamento do arquivo, recebendo como parâmetro o endereço do arquivo e sua respectiva seção a ser processada, existindo, assim, múltiplas *threads* em paralelo, processando o mesmo arquivo em diferentes seções.

PARALELIZAÇÃO EM MODO GPU

Para a paralelização do algoritmo AES no modo GPU, foram portadas as principais funções do algoritmo AES para código CUDA. As principais alterações no código incluíram adaptações de variáveis para alocação na memória da GPU e de funções criptográficas para *kernels* CUDA.

As funções para criptografia e descriptografia funcionam de forma a receber o endereço do arquivo e o índice do bloco de 16 *bytes* a ser processado; porém, no modelo de programação CUDA, *kernels* são inicializados a partir de grupos de *kernels*, quando se deve definir a quantidade de grupos a ser inicializado e a quantidade de *threads* por grupo. Essa forma de inicialização impossibilita o envio de parâmetros diferenciados para cada *thread*, logo, impossibilitando o envio do índice do bloco a ser processado por cada *thread*.

Para contornar esse problema, o índice do bloco a ser processado é encontrado a partir da seguinte fórmula:

$$\text{Índice} = ((blockIdx.\text{x} * blockDim.\text{x})*16) + (threadIdx.x * 16)$$

Nisso, *blockIdx*, *blockDim* e *threadIdx* são variáveis CUDA, indicando o ID do bloco atual do *kernel* em execução, a quantidade total de blocos inicializados, e o id da *thread* atual. Assim, é possível que cada *thread* possa calcular o índice correto do bloco a ser processado, de forma independente.

Kernels CUDA também não possuem acesso direto à memória RAM do computador, sendo necessária a alocação dos arquivos na memória global do dispositivo GPU. Após a alocação do arquivo na memória da GPU, é possível, então, a inicialização dos *kernels* adaptados para o processamento do arquivo.

Outra forma de otimização é a transferência das tabelas *S-box* da memória global da GPU para a memória compartilhada, gerando um acesso mais rápido às tabelas necessárias, por meio das *threads*. Essa transferência é feita pelo código do *kernel*, em cujo o início da execução, cada *thread* é responsável por copiar um valor de um local específico da memória global para a memória compartilhada de seu grupo.

RESULTADOS E *BENCHMARKS*

Nas seguintes seções, são analisados os resultados de performance obtidos a partir da paralelização do algoritmo AES. Os testes foram realizados a partir do tempo da execução dos processos de criptografia ou descriptografia em arquivos de diferentes tamanhos: 1KB, 1MB, 10MB, 100MB e 1000MB.

A seguir, é apresentada a configuração do computador para realização dos testes:

- Processador Core i5 750 com *clock* de 3.8GHz;
- Memória 8GB DDR3 com *clock* de 1800MHz;
- Placa de vídeo GTX680 (1536 *CUDA-cores* e *clock* dinâmico de 915MHz);
- SSD 128GB Crucial C300 (leitura sequencial de 250MB/s).

Para os testes que envolvem tempos de criptografia, os resultados representam uma média obtida a partir de três repetições sobre as operações de criptografia e três repetições sobre as operações de descriptografia com os arquivos de diferentes tamanhos. Para os tempos de transferência de arquivos, também foi realizada a média, a partir de três repetições para cada resultado.

PERFORMANCE NO PROCESSO CRIPTOGRÁFICO

Neste teste, foi analisada apenas a performance do processo criptográfico dos arquivos selecionados, ignorando-se tempos como carregamento de arquivos do HD para a memória RAM e transferência de arquivos da memória RAM principal para a memória global da GPU.

Os testes realizados mensuraram o tempo necessário para processamento pelos algoritmos de CPU *single-thread* e *multi-thread*, e GPU para arquivos de 1KB, 1MB, 10MB, 100MB e 1000MB, conforme tabela 1:

Tabela 1 - Tempo médio, em segundos, para o processamento de dados nas diferentes implementações.

Tamanho do arquivo	Tempo de processamento CPU (*Single-Thread*)	Tempo de processamento CPU (*4 Threads*)	Tempo de processamento GPU (2048 *Threads*)
1KB	*	*	*
1MB	0.015	*	*
10MB	0.093	0.046	0.016
100MB	0.922	0.281	0.078
1000MB	9.172	2.656	0.625

*Indica um valor não significativo.

A partir desses resultados, é possível notar um aumento de performance escalável em relação ao número de núcleos de um processador no algoritmo de execução em modo CPU *multi-thread* em relação ao modo *single-thread*. Para o modo CPU *multi-thread,* também foram realizados testes de performance com um número de *threads* maior que a quantidade de núcleos de CPU disponíveis no computador de teste, mas não apresentaram qualquer aumento de performance, indicando o uso completo de processamento de cada núcleo por sua respectiva *thread*.

Com esses resultados, é visto também um claro aumento de performance no algoritmo de execução em GPU em relação aos algoritmos CPU single-*thread* e *multi-thread*, obtendo uma performance até 424% mais rápida no tempo de processamento em comparação ao algoritmo de execução CPU *multi-thread*, no arquivo de 1000MB.

TEMPOS DE TRANSFERÊNCIA DE ARQUIVOS

Nessa bateria de testes, foram considerados apenas os tempos de carregamento dos arquivos do HD para a memória RAM do computador (evento realizado antes de ambos os processos de processamento na CPU e GPU) e os tempos de transferência dos arquivos da memória RAM para a memória global do dispositivo GPU, e da memória global da GPU de volta a memória RAM. Tais eventos foram efetuados antes e após os processos criptográficos dos arquivos na GPU, conforme a tabela 2.

Tabela 2 - Tempo médio, em segundos, para o carregamento e transferência de arquivos.

Tamanho do arquivo	Tempo de carregamento para a memória RAM	Tempo de transferência CPU → GPU e GPU → CPU
1KB	*	0.062
1MB	0.015	0.062
10MB	0.046	0.093
100MB	0.421	0.157
1000MB	4.392	0.922

* Indica um valor não significativo.

Com esses resultados, é possível perceber que o carregamento e transferência de arquivos se tornam os processos mais custosos entre todos os eventos para a criptografia ou descriptografia de um arquivo. Processo ligados diretamente à velocidade de leitura do HD, com uma banda efetiva de 227MB/s em um arquivo de 1000MB.

Também analisando os valores, pode-se notar que a transferência de arquivos de tamanhos pequenos (1MB ou menos) podem gerar uma quantidade grande de *overhead* na transferência de arquivos CPU → GPU, gerando, assim, uma performance geral, muito inferior para o processamento de arquivos pequenos na GPU.

Uma solução para contornar esse problema em grandes quantidades de arquivos pequenos pode ser a transferência de arquivos em blocos, carregando e transferindo diversos arquivos em uma única transferência sequencial, ignorando os custos de *overhead* de diversas micro transferências.

TEMPO TOTAL DE CRIPTOGRAFIA

Neste teste, foram levados em conta o tempo total para o processamento de um arquivo, contabilizando os tempos de leitura do arquivo a partir do HD. Nos casos de processamento via CPU nos modos *single-thread* e *multi-thread,* foram contabilizados também os tempos de processamento criptográfico dos arquivos.

No caso de processamento via GPU, além dos tempos de leitura do HD, foram contabilizados também os tempos para transferência dos dados da memória RAM do CPU para a memória global da GPU, seu tempo de processamento via GPU e seu tempo de transferência de dados da memória global da GPU para a memória RAM da CPU.

Tabela 3 - Tempo médio, em segundos, para o processo completo de criptografia.

Tamanho do arquivo	Tempo (s) CPU (*single-thread*)	Tempo (s) CPU (4 *threads*)	Tempo (s) GPU (2048 *threads*)
1KB	*	*	0.062
1MB	0.03	0.015	0.077
10MB	0.139	0.092	0.155
100MB	1.343	0.702	0.656
1000MB	13.562	7.046	5.937

* Indica um valor não significativo.

Levando em conta todos os processos para a criptografia ou descriptografia de arquivos e transferência de dados, nota-se que o algoritmo de CPU *multi-thread* apresenta uma melhora geral de desempenho em todos os tamanhos de arquivos, enquanto que o algoritmo via GPU torna-se mais eficiente apenas em arquivos de tamanho igual ou superior a 100MB. Isso se deve ao somatório dos tempos de transferência dos arquivos para a memória global da GPU e ao grande tempo de leitura dos arquivos a partir do HD, fazendo com que resultados no processamento via GPU tornem-se menos expressivos.

ANÁLISE DOS RESULTADOS

Analisando os resultados, é possível perceber um grande potencial de processamento paralelo, a partir da utilização de placas de vídeo, e a escalabilidade do algoritmo AES Rijndael em termos de paralelização, em que o algoritmo paralelizado em GPU obteve um resultado de até 424% mais velocidade de processamento em relação ao algoritmo CPU *multi-thread*.

Porém, apesar da grande capacidade de processamento, existe um notável gargalo nos processos de criptografia e descriptografia de arquivos, capaz de negativar, em parte, os ganhos de processamento via GPU, que é a velocidade de transferência de arquivos. Problema que afeta especialmente arquivos de tamanhos pequenos, que possuem baixos tempos de processamento, levando, diversas vezes, mais tempo para alocar o arquivo no dispositivo de GPU, do que o tempo para realização do processamento em si.

CONCLUSÃO

Neste trabalho, foi apresentado de que forma é realizado o processo de criptografia de dados utilizando o padrão AES, bem como a importância de sua utilização e de problemas de performance. Também foi visto que, por ser um algoritmo repetitivo e linear, é passível de paralelização com o objetivo de aumentar sua performance. Foram vistas, também, as características básicas da utilização da tecnologia CUDA, como sua hierarquia de memórias e *threads*, assim como a percepção de sua eficácia para a paralelização e otimização de algoritmos utilizando esta tecnologia.

Com o desenvolvimento deste projeto, foi criada uma análise sobre a paralelização do algoritmo AES, como dificuldades e problemas encontrados relativos à sua performance e processo de paralelização.

Porém, apesar das modificações feitas no algoritmo, ainda existem alguns pontos plausíveis de melhorias a serem estudados no futuro, como:

Realização de testes com métodos otimizados para transferência de arquivos, como o uso de funções de *zero-copy* de dados e *streaming* de eventos (agrupando chamadas de escrita e leitura na memória global);

Alocação temporária dos blocos de dados sendo processados, para a memória compartilhada da GPU ou registradores, reduzindo acessos à memória global ou à local.

REFERÊNCIAS

DAEMEN, J.; RIJMEN, V. **The Design of Rijndael**. Heindelberg: Springer, 2002.

ERDELSKY, P. J. **Rijndael Encryption Algorithm**. 2002. Disponível em: <http://bit.ly/1ONfN0E>. Acesso em: 05 maio 2015.

LUKEN, B. P.; OUYANG, M.; DESOKY, A. H. **AES and DES Encryption with GPU**. Lousville: University of Lousville, p. 67-70, 2009.

NIST. National Institute of Standards and Technology. **Advanced Encryption Standard (AES)**. 2001. Disponível em: <http://1.usa.gov/1KTor5Z>. Acesso em: 05 maio 2015.

NVIDIA. **CUDA LLVM Compiler.** 2014a. Disponível em: <http://bit.ly/1iBHfiL>. Acesso em: 01 maio 2014.

NVIDIA. **Parallel Programing and Computing Platform.** 2014b. Disponível em: <http://bit.ly/1MWjNa1>. Acesso em: 05 maio 2015.

NVIDIA. **CUDA Toolkit Documentation.** 2014c. Disponível em: <http://bit.ly/1MxkSob>. Acesso em: 05 maio 2015.