Operações Paralelas sobre Bases Massivas de Strings

Caio José dos Santos Brito[1], Lucas Gallindo Costa[3], João Marcelo X. N. Teixeira[1,2], Veronica Teichrieb[1]

[1]Voxar Labs - Centro de Informática
Universidade Federal de Pernambuco
Recife, Brasil
{cjsb, jmxnt, vt}@cin.ufpe.br

[2]Departamento de Estatística e Informática - DEINFO
Universidade Federal Rural de Pernambuco
Recife, Brasil
jmxnt@cin.ufpe.br

[3]Departamento de Eletrônica e Sistemas - DES
Universidade Federal de Pernambuco
Recife, Brasil
lucas_ra@hotmail.com

Resumo - Este trabalho analisa operações básicas comumente utilizadas em algoritmos de recuperação de dados textuais com o intuito de se beneficiar da arquitetura paralela CUDA da NVIDIA. Quatro operações diferentes foram implementadas e comparadas com suas implementações em CPU: busca exata e aproximada de strings, substituição de caracteres e cálculo de frequência de termos. Diferentes testes foram realizados variando o tamanho da base de dados, o tamanho da palavra procurada e o número de ocorrências da mesma na base. Foi possível obter uma melhora de desempenho na maioria dos cenários analisados. Uma das operações foi usada para buscar palavras no banco de dados do Diário Oficial da União, sendo possível obter um speedup de 13 vezes quando comparado com a solução online em CPU.

Palavras-chave— busca paralela de strings; GPGPU

I. INTRODUÇÃO

No início da década de 90, estudos apontavam a preferência da maioria das pessoas pela busca de informações através de outras pessoas, ao invés de utilizarem sistemas de recuperação de informação. Atualmente, as diversas otimizações na eficiência do processo de recuperação de informação levaram os motores de busca a atingir níveis de qualidade responsáveis por torná-los um padrão e a opção preferida de busca de informação. A área de pesquisa envolvendo recuperação de informação evoluiu da academia para se tornar a base do acesso à informação pela população [10].

Apesar da *Web* ter sido um dos maiores impulsionadores do acesso mais fácil à informação, a área data de antes da *web*. Algoritmos inteligentes capazes de busca e recuperação representam a tecnologia chave para atender os anseios por informação da sociedade atual. Métodos de recuperação de informação baseada em texto merecem atenção especial, principalmente devido à enorme quantidade de criadores de conteúdo presentes hoje em dia nas comunidades *Web*. O principal desafio está em acompanhar o aumento da quantidade de informação disponível e tornar a informação específica desejada cada vez mais rapidamente acessível [8]. A arquitetura inerente às GPUs (*Graphics Processing Units*), capazes de realizar em paralelo uma grande quantidade de processamento sobre dados massivos, as torna ferramenta fundamental nesse processo [7].

As GPUs foram inicialmente criadas com o objetivo de acelerar a computação gráfica, mas elas evoluíram em poderosas ferramentas capazes de serem utilizadas com propósito mais geral. Este trabalho tem como objetivo analisar algumas operações básicas utilizadas em mineração de informação baseada em texto aproveitando-se da arquitetura CUDA (*Compute Unified Device Architecture*), criada pela NVIDIA, para tal [21]. Ao todo, quatro algoritmos foram implementados em paralelo e comparados com suas implementações sequenciais de referência, em CPU: busca de padrões de caracteres, busca de padrões de caracteres com grau de similaridade, substituição de sequências de caracteres e cálculo de frequência de termos. Além do ganho de desempenho computacional obtido com as implementações paralelas, um dos algoritmos é utilizado em um estudo de caso real, demonstrando uma aplicação prática e por consequência uma contribuição direta do trabalho proposto.

O restante do artigo está organizado da seguinte forma: a seção 2 descreve trabalhos relacionados encontrados na literatura; a seção 3 detalha os algoritmos implementados usando *GPU Computing* (programação de propósito genérico em GPUs) e suas versões sequenciais em CPU; a seção 4 faz a análise dos resultados obtidos; a seção 5 descreve o estudo de caso escolhido; por fim, a seção 6 apresenta as conclusões e propõe uma continuidade para este trabalho.

II. TRABALHOS RELACIONADOS

Por constituírem a base dos algoritmos de mineração de informação baseada em texto, as operações de busca, casamento de padrões, substituição e cálculo de frequência são fundamentais para qualquer sistema mais completo de recuperação de dados. Os trabalhos relacionados listados nesta seção estão divididos em implementações (tanto em CPU como em GPU) voltadas para a busca de padrões de caracteres e em aplicações mais complexas, derivadas do uso desses algoritmos.

A. *Algoritmos de Busca de Strings*

Diversos algoritmos de busca de padrão de caracteres podem ser encontrados na literatura. Dentre os que focam em implementações sequenciais voltadas para CPU, pode-se citar os algoritmos de Knuth-Morris-Pratt [16] e Boyer-Moore [17]. Existem também algoritmos que são específicos para casos em que o padrão de caracteres a ser procurado está contido num grupo finito de elementos, como os algoritmos de Aho–Corasick [18], Commentz-Walter [19] e Rabin–Karp [20]. Nesses casos, sabe-se previamente que a base de dados a ser buscada apenas é composta por palavras específicas (conjunto finito), sendo essa informação utilizada para acelerar as buscas na mesma.

O algoritmo de Knuth-Morris-Pratt observa o erro na ocorrência de caracteres para determinar onde começar a próxima busca pela palavra. A partir dessa observação consegue-se diminuir o tempo da busca em relação ao método de força bruta implementado neste trabalho. Já no método de Boyer-Moore, ocorre um pré-processamento da *string* que está sendo procurada, mas não da base, emparelhando a palavra a partir da última ocorrência de um determinado caractere.

O trabalho descrito em [1] foi um dos pioneiros a apresentar uma solução paralela em GPU para o problema de busca de *strings*. Seus resultados mostram *speedups* de até 24 vezes em relação à implementação original em CPU. A referência [2] compara os resultados obtidos com implementações mais recentes dos algoritmos de força bruta, Boyer-Moore-Horspool e Quick-Search. São obtidos *speedups* de 36 até 106 vezes. Essa diferença de resultados entre os trabalhos propostos em [1] e [2] ocorre principalmente devido à diferença do *hardware* utilizado. Em [1] é utilizada uma GPU mais recente, com um número maior de processadores. Isso sugere uma tendência de aumento da diferença entre os desempenhos das implementações em CPU e GPU, dado que a cada nova geração de GPUs o aumento do número de núcleos é mais significativo quando comparado ao aumento de velocidade das CPUs mais recentes.

B. *Aplicações Envolvendo Algoritmos de Busca de Strings*

A busca de caracteres é base para algoritmos mais complexos, o que se verifica na aplicação da mesma em diversas áreas. Muitas delas já se beneficiam de implementações em GPU para acelerar o processamento.

Em [3] utilizam-se algoritmos de busca de caracteres, com grau de similaridade, aplicados a sequências de DNA, enquanto [4] foca na busca por sequências de proteínas.

O desempenho da aplicação de tais operações envolvendo consultas a bancos de dados é analisado em [5]. As operações básicas descritas neste trabalho são usadas para compor tarefas de avaliação de predicados, avaliações em *range* e multi-atribuição, tanto em CPU como em GPU. A implementação paralela da multi-atribuição, por exemplo, mostrou um *speedup* de até 20 vezes em relação à implementação original em CPU, mesmo levando em consideração o tempo de cópia dos dados entre CPU e GPU (e vice-versa).

Uma adaptação do *framework* MapReduce, proposto pelo Google e bastante utilizado em linguagens funcionais, é descrita em [6] e recebeu o nome de Mars. Entre os testes realizados no Mars (em GPU) estão busca de *strings*, criação de um *ranking* de páginas de *internet* e multiplicação de matrizes. Os resultados do Mars foram comparados aos do *framework* Phoenix, que representa um MapReduce distribuído em CPUs *multi-core*, apresentando um *speedup* entre 1.5 e 16 vezes para bancos de dados extensos.

Neste trabalho, uma aplicação real do algoritmo de busca de *strings* implementado em paralelo envolverá consultas textuais sobre uma base de dados pública, o Diário Oficial da União (DOU) [11]. Mais detalhes sobre o estudo de caso serão descritos na seção 5.

III. IMPLEMENTAÇÃO

Neste trabalho foram implementadas e analisadas quatro operações, a saber busca por padrões de caracteres, busca por padrões de caracteres com grau de similaridade, substituição de sequências de caracteres e cálculo da frequência de termos. As suas implementações são detalhadas na sequência, ressaltando as principais diferenças entre as versões em CPU e GPU.

A. *Busca por Padrões de Caracteres*

Nessa operação foi utilizado o algoritmo de busca *naive* (força bruta) [1]. Ele é capaz de retornar todas as ocorrências encontradas na base de caracteres, mesmo aquelas que apresentam interseção com outras palavras, conforme ilustrado na Figura 1. O desempenho do mesmo é apenas inferior ao de algoritmos mais específicos, como é o caso do Knuth-Morris-Pratt [16], o qual efetua "pulos" entre as posições analisadas no *array* de acordo com o tamanho da palavra e o Boyer-Moore [17] que efetua "pulos" de acordo com a aparição da última letra em posições anteriores à mesma.

Apesar desses algoritmos serem mais eficientes, foi escolhida a implementação do algoritmo *naive*, pois a base de dados gerada para os casos de teste são aleatórias, o que torna o algoritmo de Boyer-Moore ineficiente. Além disso, a operação de busca de padrões de caracteres com grau de similaridade, que será descrita a seguir, não depende do primeiro *mismatch* entre caracteres para dizer que a palavra não foi encontrada em uma determinada posição, o que torna o Knuth-Morris-Pratt pouco eficiente, e por isso foi implementada como uma adaptação direta da busca *naive*.

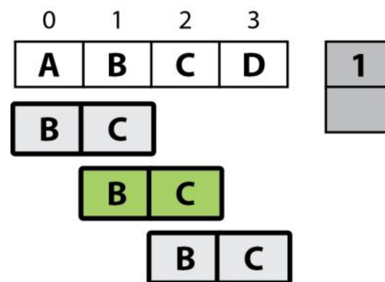

Fig. 1. Figura 1. Resultado do algoritmo de busca de padrões de caracteres implementado, quando se busca pela *string* "BC" em uma base de caracteres "ABCD". Apenas uma ocorrência é encontrada, na posição 1.

O algoritmo implementado faz uso de uma janela deslizante de tamanho igual ao da palavra a ser procurada. A cada passo,

cada caractere da palavra procurada é comparado com o caractere correspondente na base. Caso haja um caractere que não coincida, a comparação é abortada e um indicador de falha no casamento do padrão é retornado. Caso todos os caracteres sejam comparados com exatidão, um indicador de sucesso é retornado.

Na versão em GPU, cada *thread* executando em paralelo é responsável por iniciar a análise a partir de uma posição específica na base (posição de referência), e a mesma compara uma janela do tamanho da palavra a ser procurada, iniciando na posição de referência. Fez-se uso diretamente de leitura da memória global da GPU para essa operação, pois a placa de vídeo utilizada possuía otimização para esse tipo de acesso. Por isso, o uso de memória compartilhada, textura ou memória constante não trouxe ganhos ao desempenho, nesse caso.

Tanto em CPU como em GPU é possível obter a quantidade de vezes em que a palavra foi encontrada e sua posição (índice) relativa na base. A única diferença entre os resultados de CPU e GPU encontra-se no fato de que em CPU, por se tratar de um algoritmo exclusivamente sequencial, o vetor de posições retornado apresenta os endereços encontrados já ordenados. Em GPU, como diferentes *threads* executam de forma independente e em paralelo, não é possível garantir tal ordem. Caso se deseje um resultado idêntico ao de CPU, apenas se deve aplicar uma ordenação dos elementos ao final da execução do *kernel* implementado.

B. *Busca por Padrões de Caracteres com Grau de Similaridade*

Este tipo de busca considera um grau de semelhança diferente de 100% quando procura por ocorrências da palavra [9]. Todos os caracteres no espaço relativo à palavra são verificados, e é retornada uma ocorrência de palavra encontrada apenas se a quantidade de caracteres encontrados atendem ao grau de semelhança desejado, conforme ilustrado na Figura 2.

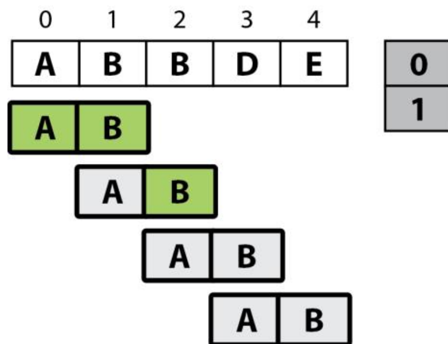

Figura 2. Resultado do algoritmo de busca de padrões de caracteres com grau de similaridade implementado, quando se busca pela *string* "AB" em uma base de caracteres "ABBDE". Duas ocorrências são encontradas, nas posições 0 e 1, respectivamente, quando se considera a tolerância de 1 caractere diferente.

C. *Substituição de Sequências de Caracteres*

Esta operação utiliza como entrada um vetor de posições, como o resultante da chamada de uma das duas operações de busca descritas anteriormente, além da palavra a ser inserida nas posições determinadas. A operação substitui todos os caracteres da palavra achada pela nova *string*. O resultado da operação de substituição de caracteres é exemplificado na Figura 3.

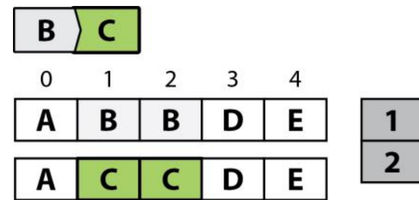

Figura 3. Resultado do algoritmo de substituição de sequências de caracteres implementado, quando se substitui o caractere "B" por "C" nas posições 1 e 2 em uma base de caracteres "ABBDE".

Na versão em CPU, para cada posição de memória encontrada, a substituição de cada caractere da palavra é feita de forma sequencial. Na GPU, por sua vez, cada *thread* é responsável por uma posição de memória no vetor de entrada e realiza a substituição da palavra nessa posição e como, diferentemente da CPU, não há garantia de ordem entres as threads, o algoritmo se torna não-determinístico em GPU. Pelo mesmo motivo que na busca por padrões, apenas a memória global da GPU foi utilizada. É importante destacar que a nível de *thread*, o processamento é realizado sequencialmente, ou seja, cada *thread* é responsável por substituir todos os caracteres de uma palavra. O que diferencia essa implementação da versão correspondente em CPU é que haverá várias *threads* executando um código sequencial ao mesmo tempo. Consequentemente, quanto maior a quantidade de *threads* em execução (nesse caso cada *thread* assume o trabalho referente a uma palavra a ser substituída), melhor o aproveitamento da GPU.

D. *Cálculo da Frequência de Termos*

Esta operação envolve uma análise mais complexa da base de caracteres, o que faz com que *threads* diferentes dependam diretamente do resultado de outras *threads*. Os algoritmos existentes para cálculo de frequência de palavras realizam a contagem das mesmas considerando um caractere-chave como separador (geralmente o caractere "espaço" é utilizado) [13]. O algoritmo implementado neste trabalho realiza um processamento mais genérico, calculando a frequência de todos os termos de quatro caracteres presentes na base, conforme ilustra a Figura 4.

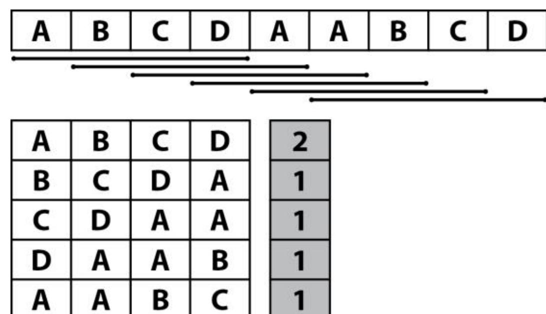

Figura 4. Resultado do algoritmo de cálculo de frequência de termos implementado, quando se utiliza uma base de caracteres "ABCDAABCD".

O cálculo de frequência é realizado através da montagem de uma tabela *hash*, onde as chaves são geradas a partir da união dos quatro caracteres do termo em um único valor inteiro sem sinal (32 *bits*). A implementação em CPU não apresenta problemas quanto à colisão de endereços na tabela *hash*, uma vez que as inserções são feitas de forma serializada. Em GPU, é necessário implementar um controle adicional para prevenir escritas simultâneas na tabela *hash*. Algumas operações fornecidas pela arquitetura CUDA, como `atomicCAS` e `atomicAdd` foram utilizadas para garantir esse controle. A primeira delas, `atomicCAS` (Compare And Set – Verifica e Atribui), é responsável por verificar se já existe algum elemento armazenado no endereço calculado na tabela *hash*. Em caso negativo, o elemento é armazenado no endereço atual. Caso já exista um elemento no endereço e ele corresponda à *hash* buscada, o valor interno na tabela é incrementado através da função `atomicAdd`, indicando mais uma ocorrência do termo encontrado. Quando a *hash* buscada não corresponde à *hash* já armazenada, um novo endereço é calculado.

O cálculo do endereço inicial de um termo baseia-se na Equação 1, onde A e B são constantes geradas aleatoriamente, p é um número primo e ST é o tamanho da tabela *hash*, conforme descrito em [12]. A função f é calculada linearmente através da multiplicação de k (valor de entrada da função de *hash*) pela constante A.

$$g(k) = (f(A,k) + B) \bmod p \bmod (ST) \quad (1)$$

IV. ANÁLISE DOS RESULTADOS

As técnicas descritas acima e implementadas em ambas plataformas, em CPU de forma sequencial e em GPU de forma paralela utilizando 1024 threads por bloco, foram comparadas a fim de verificar o ganho relativo de desempenho. Os dados referentes ao tempo de execução foram coletados a partir de um *notebook* com a seguinte configuração: Intel Core i7-3720QM 2.6 GHz, com quatro núcleos de processamento (apesar de apenas um núcleo ser utilizado) 16GB de memória RAM DDR 3, Windows 8 Professional (64 *bits*) e placa de vídeo NVIDIA GeForce GTX 680M. Em todos os testes realizados, a base de caracteres foi inicialmente gerada de forma aleatória para depois se inserir as ocorrências das palavras a serem buscadas, garantindo assim o número exato de ocorrências das mesmas. Cada teste foi repetido 10 vezes, e foi analisado o tempo médio dos mesmos.

A. Busca por Padrões de Caracteres

Três parâmetros distintos foram avaliados nos testes referentes a essa operação: variação do tamanho da base de caracteres sobre a qual a busca foi realizada; número de ocorrências da palavra buscada; e, variação do tamanho da palavra buscada.

A configuração do primeiro teste definiu como parâmetros fixos o tamanho da palavra buscada (8 caracteres) e o número de ocorrências da mesma (1024). Variou-se o tamanho da base de um milhão até um bilhão de caracteres. Ao se comparar os desempenhos de CPU e GPU, notou-se que ambas as curvas de crescimento do tempo se aproximavam de uma exponencial, sendo a de GPU com parâmetro substancialmente menor, conforme ilustrado na Figura 5. Nesse caso, obteve-se um *speedup* de 6 a 15 vezes da versão do método implementado em GPU em relação à CPU.

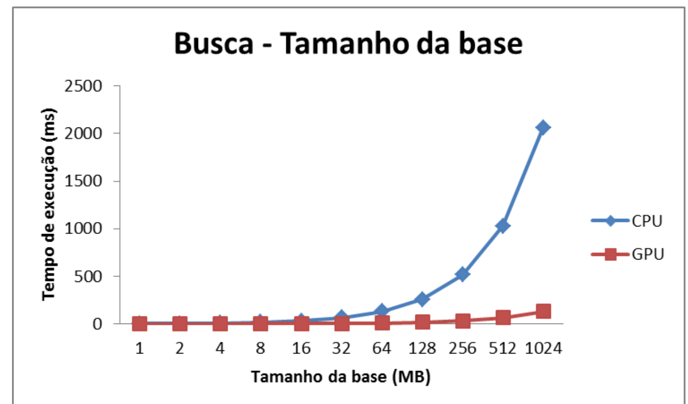

Figura 5. Comparação dos tempos de execução da operação de busca completa de caracteres variando o tamanho da base buscada.

O próximo teste envolvendo a busca completa de caracteres definiu como parâmetros fixos o tamanho da base (1 bilhão de caracteres) e o tamanho da palavra buscada (8 caracteres). Variou-se o número de ocorrências da palavra na base em potências de 2, a partir de 1 até 1024 ocorrências.

Ao se comparar os desempenhos entre CPU e GPU, percebeu-se que devido à quantidade de ocorrências testada ser relativamente pequena em comparação à quantidade de caracteres da base, não houve uma variação significativa do desempenho em cada uma das plataformas, mesmo com a variação do número de ocorrências. O fator decisivo na diferença de desempenho, nesse caso, ainda continua sendo a varredura da base de caracteres, a qual é realizada mais rapidamente em GPU devido ao elevado grau de paralelismo inerente a essa tarefa. A Figura 6 ilustra os resultados obtidos para esse teste. Pode-se observar um *speedup* médio de 16 vezes da GPU em relação à CPU.

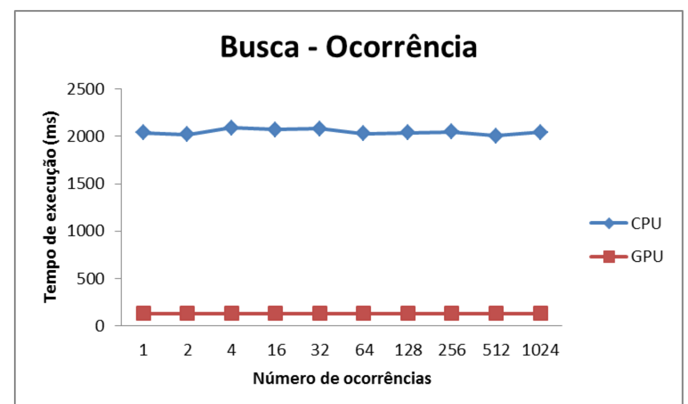

Figura 6. Comparação dos tempos de execução da operação de busca completa de caracteres variando o número de ocorrências da palavra buscada na base.

O último teste envolvendo a busca completa de caracteres definiu como parâmetros fixos o tamanho da base (1 bilhão de caracteres) e o número de ocorrências da palavra buscada

(1024). Variou-se o tamanho da palavra buscada, desde 2 até 64 caracteres. Uma vez que a busca completa pela palavra já retorna ocorrência negativa assim que encontra o primeiro caractere não correspondente, a grande maioria das *threads* apresenta o mesmo tempo de busca pela palavra, uma vez que já retornam após a verificação do primeiro caractere. Sendo assim, esse teste apresentou resultado similar ao teste anterior, conforme mostrado na Figura 7. Nela pode-se observar um *speedup* médio de 22 vezes da GPU em relação à CPU.

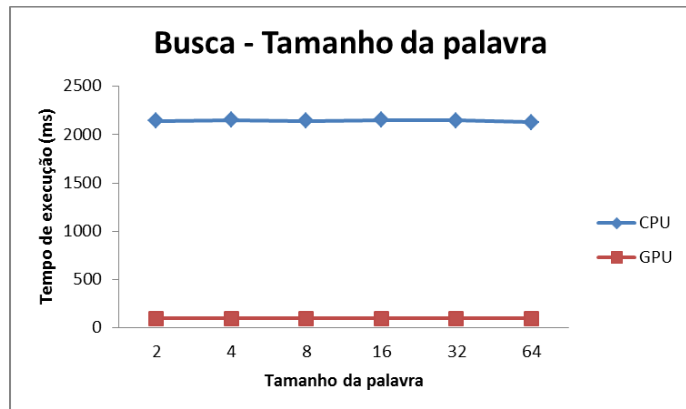

Figura 7. Comparação dos tempos de execução da operação de busca completa de caracteres variando o tamanho da palavra buscada.

B. *Busca por Padrões de Caracteres com Grau de Similaridade*

O teste realizado sobre a operação de busca por padrões de caracteres com grau de similaridade visou analisar o comportamento do desempenho da busca quando se varia o grau de similaridade. Neste trabalho, grau de similaridade reflete o número de caracteres que podem diferir da palavra original. Utilizou-se como parâmetros fixos o tamanho da base (1 bilhão de caracteres), o número de ocorrências da palavra buscada (1024) e o tamanho da mesma (16 caracteres). Também variou-se o grau de similaridade de 2 até 8, em intervalos de tamanho 2. De acordo com a Figura 8, o ganho de desempenho relativo entre GPU e CPU aumenta com o aumento do grau de similaridade da palavra buscada. Nesse caso, obteve-se um *speedup* de 10 a 13 vezes da GPU em relação à CPU.

C. *Substituição de Sequências de Caracteres*

Dois parâmetros distintos foram avaliados nos testes referentes a essa operação: variação do número de substituições da palavra buscada e variação do tamanho da palavra a ser substituída.

A configuração do primeiro teste definiu como parâmetro fixo o tamanho da palavra (8 caracteres). Variou-se o número de substituições em potências de 2, iniciando por 1 milhão até 32 milhões de ocorrências. De acordo com a Figura 9, o ganho de desempenho relativo entre GPU e CPU aumenta com o aumento da quantidade de substituições realizadas. Nesse caso, obteve-se um *speedup* de 18 a 22 vezes da GPU em relação à CPU.

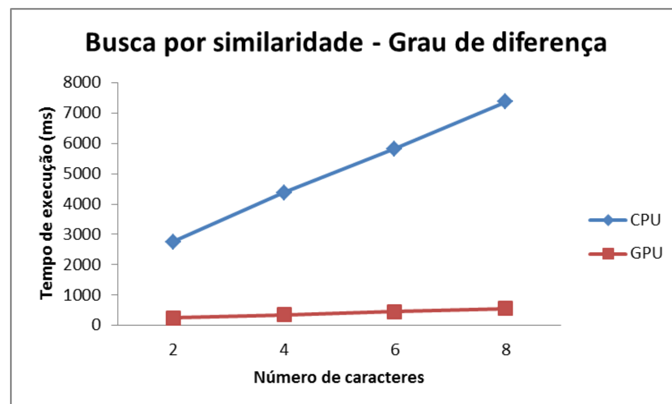

Figura 8. Comparação dos tempos de execução da operação de busca por padrões de caracteres com grau de similaridade.

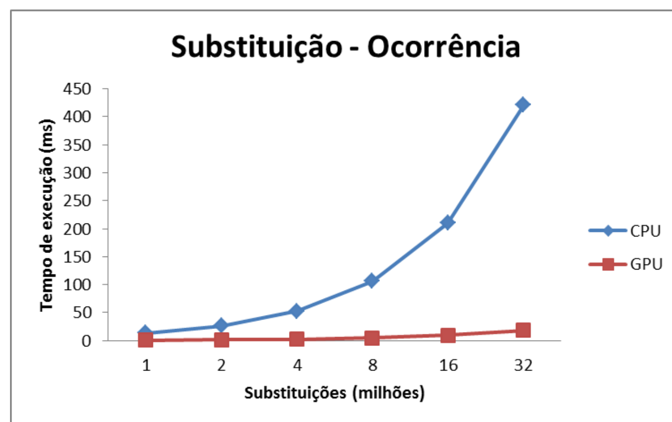

Figura 9. Comparação dos tempos de execução da operação de substituição de sequências de caracteres variando o número de substituições realizadas.

Foi realizado um segundo teste definindo como parâmetro fixo a quantidade de substituições realizadas (1 milhão) e variando o tamanho da palavra a ser substituída (desde 1 até 32 caracteres). De acordo com a Figura 10, o ganho de desempenho relativo entre GPU e CPU aumenta com o aumento da palavra sendo substituída. Nesse caso, obteve-se um *speedup* de 10 a 20 vezes da GPU em relação à CPU. Essa diferença de desempenho entre GPU e CPU para os dois testes de substituição de caracteres se justifica pelo fato da implementação realizada dividir uma escrita de memória para cada *thread*, ou seja, o número de *threads* trabalhando corresponde ao número de substituições multiplicado pelo tamanho da palavra.

D. *Cálculo da Frequência de Termos*

De todos os testes realizados neste trabalho, o cálculo da frequência de termos foi a única operação que não mostrou *speedup* em relação à implementação de referência em CPU. Para uma base com aproximadamente 786 mil caracteres, obteve-se um tempo de execução de 277.2 ms para a GPU, enquanto a CPU processou a mesma massa de dados em 143.6 ms, quase a metade do tempo da versão paralela. O desempenho obtido se deve ao fato do cálculo de frequência se basear em uma estrutura completa de tabela *hash*. De acordo com [14], a GPU não foi projetada para lidar com esse tipo de

estrutura complexa, e por isso o baixo desempenho obtido. A tabela *hash* definida em [14] foi implementada neste trabalho e mesmo processando uma quantidade maior de elementos (64 milhões), a versão em CPU ainda apresentou a metade do tempo de processamento da GPU (68.7 ms contra 140 ms).

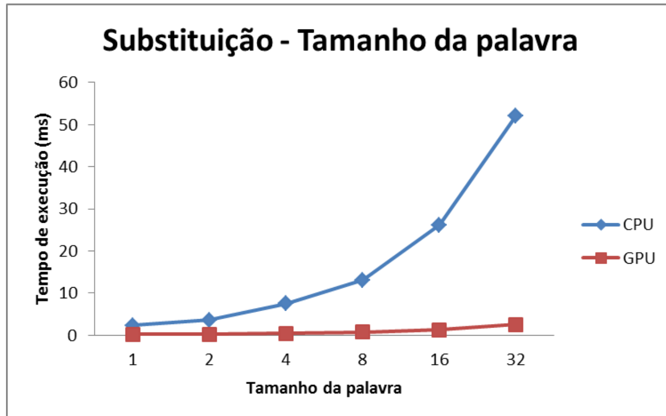

Figura 10. Comparação dos tempos de execução da operação de substituição de sequências de caracteres variando o tamanho da palavra sendo substituída.

V. Estudo de Caso

A busca por *strings* em uma base textual é uma atividade bastante comum no dia a dia das pessoas. Os *sites* de busca *online* correspondem ao maior representante desse tipo de atividade, e foi pensando nisso que esse estudo de caso foi estruturado. Inicialmente, para se comparar o desempenho da operação de busca proposta com a já existente em um *engine* de busca, é necessário possuir a mesma base de dados onde a busca será realizada. Esse motivo torna inviável comparar o desempenho da busca com o de um *site* de busca genérico, e, portanto um domínio mais específico foi escolhido: a base de dados do Diário Oficial da União [11]. O DOU é um meio oficial de comunicação através do qual a Imprensa Nacional pode tornar público todo e qualquer assunto acerca do âmbito federal. Hoje o DOU pode ser acessado virtualmente pela *internet* ou fisicamente através da compra em bancas de jornais. Para o estudo de caso em questão, foram consideradas todas as edições da primeira seção do DOU compreendidas entre 01/01/1990 e 31/12/1990, totalizando 246 edições no período selecionado. O acesso *online* retorna páginas isoladas de cada edição em formado PDF, e seu número pode variar desde 36 (menor número de páginas encontrado para uma edição) até 1208 (maior número de páginas encontrado), resultando em um total de 28,613 páginas (3,68 GB de espaço). Desenvolveu-se uma aplicação para baixar automaticamente todos os arquivos do ano selecionado, e em seguida a mesma aplicação foi responsável por converter os arquivos PDF para um único arquivo de texto (com 211 MB), o qual seria utilizado posteriormente como base para a busca. Como não se tem acesso direto ao algoritmo de busca utilizado pelo portal da Imprensa Nacional, a comparação de desempenho teve que ser realizada através de estimações, descritos de acordo com a Equação 2:

$$Tempo\ total = T_t + T_p + T_b \qquad (2)$$

Na Equação 2, T_t representa o tempo de transmissão dos dados (envio + resposta), T_p representa o tempo de processamento necessário para gerar a página *web* dado que já se tem a informação disponível e T_b representa o tempo de busca da *string* desejada na base. A estimativa do tempo de busca gasto foi realizada através do acesso a dois pontos diferentes do portal. O primeiro deles correspondeu à uma página com a lista de todas as edições dos DOUs no ano de 1990. Dessa forma, como não há busca propriamente dita, e apenas uma listagem dos arquivos no servidor, o tempo de busca foi desconsiderado. Essa página foi acessada 100 vezes e guardou-se o menor tempo total obtido, que foi de T_1 = 208 ms ($T_t + T_p$). Em seguida, o segundo acesso correspondeu à busca do termo "SUS" para o mesmo período selecionado. Após realizar a mesma busca 100 vezes, o menor tempo total obtido, T_2 = 584 ms ($T_t + T_p + T_b$), foi armazenado. Realizando a diferença entre os tempos T_2 e T_1, é possível obter uma estimativa para o tempo de busca do termo na base, que foi de aproximadamente 376 ms. Analisando o tempo médio da mesma operação de busca completa implementada em GPU, obteve-se 28 ms. Esse valor indica um *speedup* de 13 vezes em relação ao algoritmo utilizado pelo portal da Imprensa Nacional, ratificando o ganho que se pode ter ao utilizar o poder de processamento das GPUs para acelerar buscas em bases de dados, conforme descrito em [15].

VI. Conclusão

Este trabalho realizou um estudo comparativo entre diferentes operações sobre cadeias de caracteres implementadas tanto em CPU como em GPU: busca por padrões de caracteres, busca por grau de similaridade, substituição de sequências de caracteres e cálculo da frequência de termos. As bases de caracteres utilizadas neste trabalho compreenderam sequências geradas aleatoriamente assim como bases de texto reais, como textos extraídos de PDFs do Diário Oficial da União.

Na busca por padrões de caracteres foi possível obter um *speedup* de no mínimo 18 vezes, na busca por grau de similaridade foi possível obter uma melhora de desempenho de no mínimo 10 vezes e, por fim, na substituição de caracteres foi encontrado um *speedup* entre 18 a 22 vezes. Para operação de cálculo de frequência de termos não foi possível obter *speedup*, uma vez que a tabela *hash* implementada apresentou melhor desempenho em CPU, devido à complexidade inerente ao algoritmo.

Para o estudo de caso real, foi utilizado um subconjunto da base de dados do Diário Oficial da União do ano de 1990. Comparando a busca da palavra "SUS" foi possível verificar um *speedup* de 13 vezes em relação à solução atual.

A maioria dos resultados mostrou uma clara melhora de desempenho quando se usa a GPU como alternativa à CPU, comprovando que para banco de dados massivos o uso da GPU melhora o desempenho em aproximadamente 15 vezes, até mesmo se compararmos com aplicações *web*.

Como trabalhos futuros, pretende-se realizar novos testes em ambientes não controlados usando bases de caracteres diferentes, além de aumentar a escalabilidade da solução proposta através do uso de multi-GPU.

AGRADECIMENTOS

Os autores gostariam de agradecer ao CNPq por financiar parcialmente esta pesquisa (projeto Propesq PIBIC nº 13015479).

REFERÊNCIAS

[1] Charalampos S Kouzinopoulos and Konstantinos G Margaritis. String matching on a multicore gpu using cuda. In *Informatics, 2009. PCI'09. 13th Panhellenic Conference on*, pages 14–18. IEEE, 2009.

[2] Raymond Tay. Demonstration of Exact String Matching Algorithms using CUDA. Unpublished work.

[3] Yu Liu, Longjiang Guo, Jinbao Li, Meirui Ren, and Keqin Li. Parallel algorithms for approximate string matching with k mismatches on cuda. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International, pages 2414–2422. IEEE,* 2012.

[4] Svetlin A Manavski and Giorgio Valle. Cuda compatible gpu cards as efficient hardware accelerators for smith-waterman sequence alignment. *BMC bioinformatics*, 9(Suppl 2):S10, 2008.

[5] Naga K Govindaraju, Brandon Lloyd, Wei Wang, Ming Lin, and Dinesh Manocha. Fast computation of database operations using graphics processors. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data,* pages 215–226. ACM, 2004.

[6] Wenbin Fang, Bingsheng He, Qiong Luo, and Naga K Govindaraju. Mars: Accelerating mapreduce with graphics processors. *Parallel and Distributed Systems, IEEE Transactions on*, 22(4):608–620, 2011.

[7] Peter Wittek and S´andor Dar´anyi. Accelerating text mining workloads in a mapreduce-based dis tributed gpu environment. *Journal of Parallel and Distributed Computing*, 2012.

[8] Donald E Knuth, James H Morris, Jr, and Vaughan R Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977.

[9] Jorma Tarhio and Esko Ukkonen. Boyer-moore approach to approximate string matching. In *SWAT 90,* pages 348–359. Springer, 1990.

[10] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to information retrieval,* volume 1. Cambridge University Press Cambridge, 2008.

[11] Imprensa Nacional. Diário oficial da união, July 2013.

[12] Dan A Alcantara, Andrei Sharf, Fatemeh Abbasinejad, Shubhabrata Sengupta, Michael Mitzenmacher, John D Owens, and Nina Amenta. Real-time parallel hashing on the gpu. In *ACM Transactions on Graphics (TOG),* volume 28, page 154. ACM, 2009.

[13] TR Lynam, CLA Clarke, and GV Cormack. Information extraction with term frequencies. In *Proceedings of the first international conference on Human language technology research*, pages 1–4. Association for Computational Linguistics, 2001.

[14] Jason Sanders and Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming.* Addison-Wesley Professional, 2010.

[15] Peter Bakkum and Kevin Skadron. Accelerating sql database operations on a gpu with cuda. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, pages 94–103. ACM, 2010.

[16] Knuth, Donald E.; Morris, JR, James H.; Pratt, Vaughan R. Fast pattern matching in strings. SIAM journal on computing, v. 6, n. 2, p. 323-350, 1977.

[17] Boyer, Robert S.; Moore, J. Strother. A fast string searching algorithm. Communications of the ACM, v. 20, n. 10, p. 762-772, 1977.

[18] Aho, Alfred V.; Corasick, Margaret J. Efficient string matching: an aid to bibliographic search. Communications of the ACM, v. 18, n. 6, p. 333-340, 1975.

[19] Commentz-Walter, Beate. A string matching algorithm fast on the average. Springer Berlin Heidelberg, 1979.

[20] Karp, Richard M.; Rabin, Michael O. Efficient randomized pattern-matching algorithms. IBM Journal of Research and Development, v. 31, n. 2, p. 249-260, 1987.

[21] NVIDIA CUDA Compute Unified Device Architecture - CUDA C Programming Guide, 2012