String matching aproximado: uma abordagem thread-cooperative em GPU do algoritmo de Myers

Ygor Canalli[1], **Alexsander A. Melo**[1], **Marcelo P. Zamith**[1]

[1] Departamento de Ciência da Computação
Universidade Federal Rural do Rio de Janeiro (Nova Iguaçu – RJ).

{ygorcanalli,alexsandermelo,zamith}@ufrrj.br

1. Introdução

O *string matching aproximado* é um problema clássico em ciência da computação, o qual possui aplicações em diversas áreas, como em: correção ortográfica, bioinformática e processamento de sinais [Hyyrö 2002]. Dado um texto T de tamanho m, um padrão p de tamanho n, e um limiar $\ell \geq 0$, busca-se uma substring t de T tal que a *distância de edição* (*i.e.* a *distância de Levenshtein*) entre p e t é no máximo ℓ [Hyyrö 2002]. O algoritmo original para cálculo da distância de edição faz uso da técnica de programação dinâmica e possui complexidade de tempo e espaço $\mathcal{O}(nm)$. Com o intuito de otimizar tal computação, Myers propôs um algoritmo para string matching aproximado com complexidade de tempo $\mathcal{O}(nm/w)$ e espaço $\mathcal{O}(nm|\Sigma|/w)$, para $n \leq w$, obtido através do uso do paralelismo no nível de bits aplicado ao cálculo da distância de edição, onde w é o tamanho da palavra da máquina e $|\Sigma|$ é o tamanho do alfabeto utilizado [Myers 1999].

Em [Chacón et al. 2014], são apresentadas duas abordagens de paralelismo em GPU do algoritmo de Myers. A primeira é a *task-parallel*, cujo objetivo é paralelizar a busca de diferentes padrões para um mesmo texto. E a outra abordagem é a *thread-cooperative*, onde objetiva-se realizar a busca de um único padrão grande ($n > w$) num único texto, através da divisão do padrão em fatias de tamanho no máximo w. No entanto, observe que proveniente das propagações das operações de soma e *shift-left* inerentes ao algoritmo de Myers, há uma alta dependência de dados entre as fatias de um padrão, o que possivelmente prejudica esta estratégia de paralelismo, apesar dos bons resultados obtidos por [Chacón et al. 2014]. Além disso, note que geralmente $n \ll m$. Desta forma, propomos uma estratégia *thread-cooperative* em GPU que, em vez do padrão, divide o texto em diversas fatias, onde cada *thread* fica responsável por realizar a busca de um mesmo padrão em uma das fatias do texto, evitando-se assim dependência de dados.

2. Metodologia e Resultados

Sob essa estratégia de dividir o texto de entrada em diversas fatias, um bom número de *threads* a serem lançadas seria intuitivamente $\lfloor m/n \rfloor$, o que corresponderia a uma divisão balanceada do texto, na qual cada *thread* ficaria responsável por exatamente uma fatia. Entretanto, como um *warp* pode executar até 32 *threads* por vez, é interessante que a quantidade de *threads* a serem lançadas seja sempre potência de dois. Assim, lançamos 2^q *threads*, onde $2^q \leq \lfloor m/n \rfloor < 2^{q+1}$ e $q \in \mathbb{N}$. Ademais, observe que possivelmente $r = m \bmod 2^q \neq 0$. E como desejamos dividir as cargas entre as *threads* da forma mais equilibrada possível, acrescemos à responsabilidade das r primeiras *threads* uma posição adicional do texto. Por fim, observe que com essa estratégia de divisão do texto, surge o problema de *bordas* de *string-matching*, onde ocorrências do padrão próximas das bordas

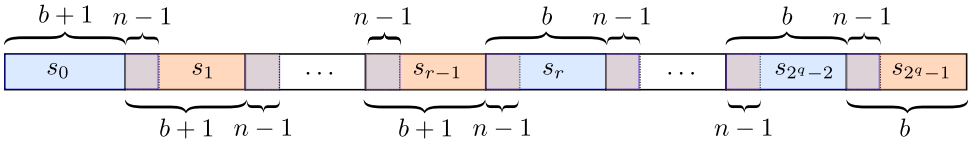

Figure 1. Divisão de carga entre as *threads*, onde s_i denota a i-ésima fatia.

de duas fatias não são detectadas. Para contornar este problema, cada *thread* (exceto a de maior id) ficará encarregada por, além de verificar sua respectiva fatia do texto, de tamanho $b = \lfloor m/2^q \rfloor$, buscar a ocorrência do padrão em uma área de sobreposição de tamanho $n-1$ da fatia subsequente. Figura 1 ilustra essa divisão de carga entre *threads*.

Resultados. Em nossos experimentos utilizamos uma placa gráfica NVIDIA GT 740M com 384 *cuda cores* e um processador Intel Core i5 4200U com dois *cores*; a base de dados utilizada em nossos testes foi a *Bíblia King James*. Comparamos o desempenho da nossa abordagem *thread-cooperative* em GPU com a execução sequencial em CPU do algoritmo de Myers. A Tabela 1 apresenta alguns resultados preliminares que obtivemos. Uma observação importante a se fazer é a de que padrões maiores fazem com

n	m	CPU Sequencial (ms)	Nossa abordagem (ms)	*Speed-up*
4	13 369	1,269000	0,037600	33,75
4	166 659	6,247000	0,298561	20,92
4	240 885	8,427000	0,414089	20,35
31	8 703 687	279,370026	115,873180	17,60
31	166 659	6,340000	0,390404	16,23
31	240 885	8,577001	0,546658	15,68

Table 1. Análise comparativa de desempenho.

que menos *threads* sejam lançadas, o que prejudica o desempenho, conforme podemos verificar nos experimentos. Um outro fator prejudicial ao desempenho, foi o acesso divergente à memória. Todavia, diante dos resultados obtidos, podemos ver que, apesar desses fatores prejudiciais, a nossa estratégia de paralelismo obteve ganhos consideráveis de desempenho se comparado ao tempo de execução em CPU, de até 3 375% de melhora.

3. Considerações finais e Trabalhos futuros

Baseado nos resultados, percebe-se que a estratégia proposta é uma abordagem promissora de paralelismo em GPU do algoritmo de Myers, valendo assim o esforço de um estudo posterior mais minucioso. Como trabalhos futuros, objetivamos realizar otimizações de acesso à memória e realizar testes mais conclusivos, comparando a nossa abordagem com a abordagem proposta por [Chacón et al. 2014].

Referências

Chacón, A., Marco-Sola, S., Espinosa, A., Ribeca, P., and Moure, J. C. (2014). Thread-cooperative, bit-parallel computation of levenshtein distance on gpu. In *Proceedings of the 28th ACM International Conference on Supercomputing*, ICS '14, pages 103–112, New York, NY, USA. ACM.

Hyyrö, H. (2002). Bit-parallel approximate string matching algorithms with transposition. In *In Proc. 13th Combinatorial Pattern Matching (CPM 2002), LNCS 2373*, pages 203–224.

Myers, G. (1999). A fast bit-vector algorithm for approximate string matching based on dynamic programming. *J. ACM*, 46(3):395–415.