Paralelismo na busca por Padrões Textuais

Cristiano A. Künas[1], **Leandro P. Heck**[1],
Douglas E. Hoffmann[1], **Edson L. Padoin**[1]

[1] Departamento de Ciências Exatas e Engenharias (DCEENG)
Universidade Regional do Noroeste do Estado do Rio Grande do Sul – (UNIJUÍ)
Santa Rosa – RS – Brasil

cristianokunas@outlook.com, leandroph1992@hotmail.com,
douglas22.hoffmann@hotmail.com, padoin@unijui.edu.br

Resumo. *O presente artigo apresenta uma análise de desempenho da paralelização de uma aplicação de busca por padrões textuais. O objetivo do trabalho é avaliar a utilização da biblioteca* MPI *na paralelização de aplicações reais utilizadas em pesquisas de strings em arquivos de tamanho grandes. Os resultados obtidos apresentaram speed-up de até* $2,8$ *vezes se comparado com a versão sequencial utilizando um processador convencional.*

1. Introdução

A técnica de análise de padrões em textos é uma área de pesquisa que vem sendo estudada por parte de vários pesquisadores da área de ciência da computação [de Ávila and Soares 2012]. Objetiva-se definir se duas ou mais instâncias de dados (strings, árvores, tuplas e outros) reproduzem o mesmo objeto do mundo real [Da Silva et al. 2007].

A correspondência de sequências é essencial em muitas aplicações e disciplinas, como processamento de sinais, análise de texto, autenticação e verificação, análise e reconhecimento de fala, recuperação de informações, sequenciamento de DNA e análise forense, compactação de dados e ciências computacionais [Kabir et al. 2014]. Diante deste problema é que algoritmos paralelos são desenvolvidos almejando um melhor desempenho e uso eficiente do poder computacional.

Este trabalho tem por objetivo realizar uma análise da viabilidade de utilização de MPI na paralelização de aplicações de busca por padrões textuais. Por meio desta ferramenta é possível dividir o problema em partes menores, que são executadas paralelamente em vários processadores.

O restante do trabalho esta organizado da seguinte forma. A Seção 2 discute os trabalhos relacionados. A Seção 3 apresenta a metodologia utilizada na implementação e o ambiente de execução utilizado na realização dos testes. Na Seção 4 são discutidos os resultados, seguidos das Conclusões e Trabalhos Futuros.

2. Trabalhos Relacionados

Na literatura diversos trabalhos apresentam estudos sobre busca de padrões em textos. Brođanac, Budin e Jakobović (2011) apresentam um trabalho que utiliza o algoritmo Rabin-Karp e uma maneira de paralelizá-lo, melhorando o seu desempenho em sistemas multiprocessados de descoberta de sequencia exata e todas as ocorrências de uma *string*

em outra *string*. Um dos segmentos da ciência onde este tipo de pesquisa é aplicada em um nível substancial é a biologia, especialmente no segmento sobre as cadeias de DNA.

Um sistema de anotação semiautomático para habilitar pesquisa semântica é apresentado por Liu em [Liu et al. 2009], que captura os dados textuais web usando o algoritmo Knuth-Morris-Pratt (KMP). A solução proposta visa obter melhores resultados nas pesquisas textuais, uma vez que a notação semântica reduz a quantidade de termos inadequados obtidos. Os resultados alcançados apresentaram uma melhora significativa nas buscas realizadas em sistemas como o Google, Yahoo e Bing em comparação ao simples uso de palavras-chaves.

Esses trabalhos contemplam importantes contribuições no uso de algoritmos de comparação e busca em textos, porém nenhum destes aborda a solução proposta no presente artigo. Para este trabalho, tendo como base o algoritmo sequencial padrão de busca textual, foi implementada uma versão paralela utilizando a biblioteca MPI, visando obter melhor desempenho.

3. Metodologia

Para execução paralela, fez-se um estudo do algoritmo sequencial, e com base neste foi desenvolvido a nova versão com o uso do MPI. Este estudo foi realizado almejando melhorar os tempos de execução das simulações em busca de padrões em texto.

O algoritmo sequencial possui dois métodos, TopDown e Bottom-Up. O primeiro realiza a busca a partir do início do arquivo de texto, já o segundo, faz o inverso. Diferente da versão sequencial, na versão paralela implementada, a busca Bottom-Up foi modificada empregando um método que inverte a *string* pesquisada. Com esta meloria foi possível utilizar somente um método para todas as buscas (checkTextForPatternAtPosition), na qual abre o arquivo e o lê dentro do intervalo passado como parâmetro, sempre do início para o final do arquivo texto.

Um dos grandes desafios computacionais da atualidade se concentra em manipular e analisar de forma ágil e com baixo custo computacional, um volume grande de dados [Matteussi et al. 2017]. Por esta razão, foi escolhido utilizar a biblioteca MPI, onde é possível dividir o trabalho entre vários processadores, agilizando o processo de busca.

Para a análise de desempenho, buscou-se a variação no número de processos bem como o tamanho do arquivo texto. Foram realizados testes de 1 até 10 processos, utilizando arquivos com tamanhos de 25, 50 e 100 MB. Foram realizados 10 testes obtendo-se uma média aritmética e *speed-up*. Foram definidos dois métodos, *master* e *slave*, onde *master* é responsável pela divisão do trabalho e cada *slave*, por realizar a tarefa de busca na sua parte do arquivo.

No método *master* são declaradas as variáveis *count, remainder, start e stop*. *Count* receberá a divisão inteira entre o tamanho do texto e o número de escravos. *Remainder* irá receber o resto da divisão entre o tamanho do texto e o número de escravos. Para cada escravo é atribuído o ponto de início e de parada através de um laço de repetição *for*, e realizado o envio com os métodos do MPI .

O método *MPI_Send()* faz o envio das informações para cada escravo. Em cada escravo, o método *MPI_Recv* faz a recepção das informações. Após, estas informações

são passadas ao método *checkTextForPatternAtPosition*, que faz a analise da sequência de caracteres e realiza a busca dentro do texto.

3.1. Ambiente de Execução

Para a execução foi utilizado um processador Intel(R) Core(TM) i5-4210U. O processador possui 2 cores e 4 threads com 1.70 GHz de frequência. Este equipamento também possui 8 GB de memória RAM DDR3 de frequência 1600 MHz. Para os experimentos, utilizou-se o sistema operacional Ubuntu versão 18.04.1 LTS, kernel $4.15.0 - 38 - generic$. A versão do MPI foi 1.6.5 e do compilador g++ a versão 7.3.0.

4. Resultados

O tempo médio de execuções da aplicação sequencial foi de $19,60 \pm 0,32$ s, $40,09 \pm 0,30$ s e $80,75 \pm 0,49$ s quando utilizado arquivo de texto de tamanho 25 MB, 50 MB e 100 MB respectivamente (Figura 1). Observa-se um *overhead* de $1,69$ % no tempo de execução quando executado a versão paralela com 2 processos. Este aumento no tempo, foi percebido nos testes com todos os tamanhos testados e é resultado da divisão do trabalho realizada pelo processo *master*.

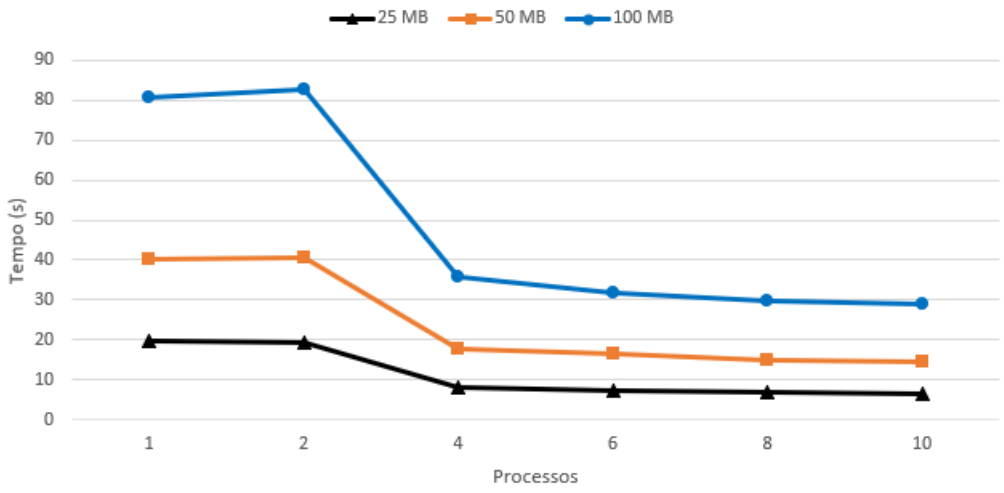

Figura 1. Tempo de execução (s) das simulações com diferentes tamanhos de arquivo.

A versão paralela executado com mais de 3 processos, tem-se *speed-up* significativos onde o *overhead* é compensado com a redução do tempo total de execução. Para arquivo de tamanho 50 MB o tempo médio de execução foi de $23,98 \pm 0,47$ s, que representa um *speed-up* de $1,67$.

O maior *speed-up* para os arquivos de tamanho 25 MB e 50 MB (Figura 1) foram alcançados todos com 10 processos. Nos experimentos, o tempo de execução foi reduzido de $19,60$ s para $6,66 \pm 0,11$ s e de $40,09$ s para $14,50 \pm 0,26$ s, o qual representa ganhos de até $2,94$ vezes e $2,76$ vezes respectivamente sobre o algoritmo sequencial.

Quando dobrado o tamanho do arquivo (de 50 para 100 MB), o tempo de execução da versão sequencial também dobra (de $40,09 \pm 0,30$ s para $80,74 \pm 0,49$ s). Entretanto,

a redução no tempo total de execução também foi proporcional, ou seja, para arquivo de tamanho 100 MB tem-se *speed-up* de até $2,80$ (Figura 1), obtendo um tempo médio de $28,81 \pm 0,13$ s.

5. Conclusão

Com base nos testes realizados, percebe-se que o tempo de execução da aplicação tem aumento proporcional ao tamanho do arquivo texto. Nota-se também, que os ganhos alcançados com a versão paralela, *speed-up* , foram semelhante nas execuções paralelas independente do tamanho do arquivo de texto utilizado. Fazendo uma projeção, se utilizado um arquivo de 100 GB, a versão sequencial levaria $22,43$ horas para ser executada. Com a versão paralela implementada, executando 10 processos, o tempo seria reduzido para 8 horas.

Como trabalhos futuros pretende-se executar a aplicação de modo que seja possível utilizar os recursos de memória compartilhada e distribuída (OpenMP + MPI), distribuindo o processamento entre vários processadores multi-core. Também pretende-se utilizar arquivos com dados reais e tamanhos maiores. Outra alternativa prevista é a execução utilizando sistemas computacionais que disponham de unidades de processamento com GPUs.

Referências

Brođanac, P., Budin, L., and Jakobović, D. (2011). Parallelized rabin-karp method for exact string matching. In *Information Technology Interfaces (ITI), Proceedings of the ITI 2011 33rd International Conference on*, pages 585–590. IEEE.

Da Silva, R., Stasiu, R., Orengo, V. M., and Heuser, C. A. (2007). Measuring quality of similarity functions in approximate data matching. *Journal of Informetrics*, 1(1):35–46.

de Ávila, R. L. and Soares, J. M. (2012). Concepção de ferramenta de apoio à correção de questões dissertativas com base na adaptação de algoritmos de comparação e busca textual combinados com técnicas de préprocessamento de textos. *RENOTE*, 10(3).

Kabir, M. N., Alginahi, Y. M., and Tayan, O. (2014). Efficient search of a sequence of words in a large text file. In *2014 World Symposium on Computer Applications Research (WSCAR)*, pages 1–6.

Liu, C.-H., Chen, H.-C., Jain, J.-L., and Chen, J.-Y. (2009). Semi-automatic annotation system for owl-based semantic search. In *Complex, Intelligent and Software Intensive Systems, 2009. CISIS'09. International Conference on*, pages 475–480. IEEE.

Matteussi, K. J., de Souza Jr, P. R., dos Anjos, J. C., and Geyer, C. F. (2017). *Um Guia Para a Modelagem e Desenvolvimento de Aplicações Big Data Sobre o Modelo de Arquitetura Zeta*. ERAD 2017.