I Encontro de Iniciação Científica e Tecnológica
I EnICT
IFSP – Câmpus Araraquara
21 de Outubro de 2016

PROCESSAMENTO PARALELO APLICADO EM ESTRUTURAS DE DADOS OTIMIZADAS COM INSTRUÇÕES ATÔMICAS

GABRIEL MELO COSTA[1], ANDRÉ DE SOUZA TARALLO[2]

[1] Graduando em Análise e Desenvolvimento de Sistemas, Bolsista PIBIFSP, IFSP Campus Boituva, mellogab83@gmail.com
[2] Docente, Departamento de Computação, IFSP Campus Araraquara, andre.tarallo@ifsp.edu.br

Área de conhecimento (Tabela CNPq): Arquitetura de Sistemas de Computação – 1.03.04.02-9

RESUMO: A informação é um conjunto de dados organizados, que tem grande importância, especialmente quando se trata de informação digital. As informações digitais podem ser transmitidas ou processadas, o que exige tecnologia disponível e custo computacional para a execução dessas tarefas. Este artigo apresenta um algoritmo para a inserção e pesquisa de uma grande quantidade de dados utilizando uma fila não bloqueante, com base em um vetor. Os métodos de inserção e remoção são baseados em instruções atômicas load link (LL) e store condicional (SC), que evitam problemas de processamento, associados com várias inserções e remoções que podem ocorrer simultaneamente. Os testes de complexidade ciclomática obtidos geraram resultados que variam entre 3 e 11 em uma escala que varia de 0 a 20, indicando facilidade de manutenção do algoritmo. Nos testes de tempo de execução, para a função de inserção foi obtido um mesmo tempo de 0,0160 segundos para inserir 20.000. 40.000 e 100.000 registros; para a função de busca, os tempos obtidos foram de 2,25 segundos, 4,67 segundos e 11,50 segundos para 20.000, 40.000 e 100.000 registros respectivamente. Conclui-se que o algoritmo tem potencial para a continuação de novas pesquisas e testes de desempenho, que visam hardware para o consumidor.
PALAVRAS-CHAVE: algoritmo não bloqueante; estrutura de dados; *load link* (LL); otimização de desempenho; *store condicional* (SC).

INTRODUÇÃO

Cada vez mais o ser humano trabalha com diversas tarefas simultaneamente, sejam elas pessoais ou profissionais, visando aumentar a produtividade no trabalho e diminuir o tempo de execução de tarefas. Na área computacional, este conceito aplica-se da mesma maneira, relacionado ao processamento paralelo (NAVAUX, 1988). A ideia de processamento paralelo data de 1920, quando Vanevar Bush, um engenheiro do MIT, apresentou um computador capaz de resolver equações diferenciais em paralelo. Von Neumann em 1940, apresentou um *grid* para a resolução do mesmo problema. (NAVAUX, 1988). O processamento paralelo é uma técnica eficiente no processamento de informações com foco na exploração de eventos concorrentes na área computacional. Neste trabalho será apresentado uma estrutura de dados não bloqueante com o objetivo de processar dados em grande quantidade, utilizando técnicas não bloqueantes, como as instruções atômicas *load link (LL)* e *store condicional (SC)*, que garantem que a execução de um *thread* não influencie no processamento de outra, funcionando de forma eficiente e organizada.

O processamento de um valor por um thread que já tenha sido processado por outro *thread*, pode implicar em erros graves no algoritmo e, portanto, no resultado final do mesmo. Este problema é chamado de 'Problema ABA' (EVEQUOZ, 2008). De acordo com as implicações citadas anteriormente, será proposto neste artigo o desenvolvimento de um algoritmo livre de bloqueio, utilizando técnicas para minimização do problema ABA, com o objetivo de otimizar o acesso aos dados.

METODOLOGIA

Baseado nas instruções LL e SC, pode-se desenvolver uma fila em linguagem C isenta ao Problema ABA, com as funções de inserção e remoção dependentes do resultado das funções LL e SC. Uma operação LL funciona basicamente como um carregador de endereços de memória, carregando o valor da variável no registrador, devolvendo como parâmetro o endereço daquela variável salva. A principal diferença da instrução SC é que ela só atualiza o valor armazenado, se nenhum outro *thread* em execução tenha alterado o valor anteriormente (ARPAC, 2015). Caso o procedimento tenha ocorrido normalmente, o valor é atualizado e devolvido como

parâmetro de retorno, com o valor inteiro 1. Caso contrário, o valor não é atualizado e o valor inteiro de retorno é 0 (ARPAC, 2015).

A fila é implementada no algoritmo com um vetor do tipo **node*, que pode ou não possuir tamanho fixo, e usa as variáveis *head* e *tail* para marcar o início e final da fila. Um nó do vetor é composto por um ponteiro que indica qual o próximo nó e um campo para armazenar dados. A variável *head* indica qual o primeiro nó da fila que pode possuir um item armazenado ou estar vazio. Já a variável *tail* indica o último nó da fila, que também pode possuir um item armazenado ou estar vazio, indicando a disponibilidade de uso.

Para inserir um valor na fila, primeiro é verificado se a fila está cheia usando como parâmetros a variável *Head* e o tamanho da fila. Caso a fila não esteja cheia, a operação de inserção lê o primeiro valor da variável *Tail* e armazena na variável *slot* o endereço de memória do final da fila. O segundo teste verifica se o nó reservado para a inserção ainda corresponde ao final da fila, ou seja, se ele ainda está vazio e pronto para inserção. A finalidade desta segunda condição é evitar que um *thread* manipule um nó que já foi utilizado por outro *thread*, causando erros no resultado final do algoritmo. Caso esteja vazio, são executadas as instruções LL/SC, responsáveis pela leitura do endereço no qual será armazenado o valor e também responsáveis pelo armazenamento deste endereço de memória. Para verificar o desempenho do algoritmo implementado, utilizou-se métricas de desempenho e análise semântica, como a Complexidade Ciclomática, LOC, eLOC e lLOC (QUICOLI, 2016).

A Complexidade Ciclomática é definida como uma métrica utilizada para analisar quão difícil será testar, manter ou realizar alterações em um código fonte, bem como apontar possíveis erros que o código venha a produzir (QUICOLI, 2016). As métricas LOC, eLOC e lLOC contabilizam as linhas de código, comentários e espaços em branco. O objetivo destas métricas é estimar a produtividade ou a capacidade de manutenção de um programa já implementado (BROOKS, 2008). Segundo Mccabe, autor da Complexidade Ciclomática, quanto menor o resultado mais fácil de manter e alterar o código fonte (MCCABE, 1996). Os resultados do teste da Complexidade Ciclomática são obtidos por meio de uma pontuação própria, geradas sobre uma análise da complexidade do código (QUICOLI, 2016).

RESULTADOS E DISCUSSÃO

O gráfico da Figura 1 ilustra os resultados obtidos por meio da análise de Complexidade Ciclomática sobre as funções LL, SC, Inserção, Remoção e Exibir. As barras azuis representam a Complexidade Ciclomática que é composta por parâmetros, retornos e estruturas condicionais utilizadas no algoritmo ou método analisado. As barras laranjas representam a análise semântica que se baseiam nas métricas LOC, eLOC, lLOC. Os resultados variaram entre 3 e 11 em uma escala que varia de 0 a 20, em que resultados até 12/13 são considerados algoritmos bem estruturados com fácil manutenibilidade.

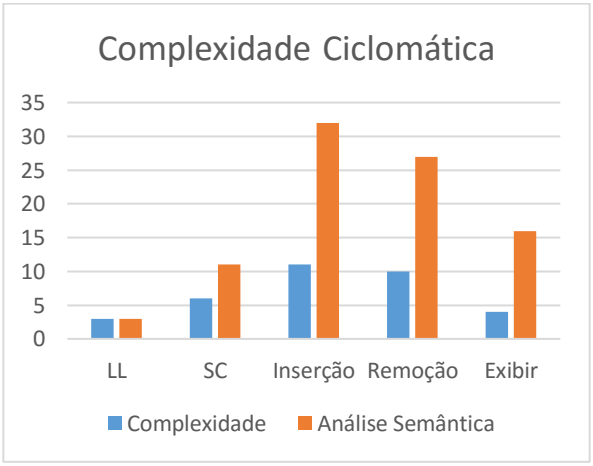

Figura 1 - Resultados da Análise de Complexidade Ciclomática.

Foram realizados testes de tempo de execução para o método de inserção com a quantidade de registros que variam de 20.000 a 100.000 registros. Em todos os testes o tempo apresentado no método de inserção foi de 0.160 segundos.

Considerando o cenário de busca de dados, os resultados são mais expressivos. Os parâmetros para os testes são os mesmos dos testes realizados nas inserções: 20.000, 40.000 e 100.000 registros. Os resultados

obtidos são: 2,25; 4,67 e 11,50 segundos, respectivamente (Figura 2). Analisando os testes nas funções de inserção e busca, nota-se uma diferença de tempo de execução, mesmo utilizando a mesma quantidade de dados inseridos nas duas funções. A diferença é que a função de busca realiza instruções que demandam mais recursos e consequentemente mais tempo, como verificar se o dado existe naquela posição e mostrá-lo na tela, enquanto que a função de inserção precisa apenas, necessariamente, inserir um valor repassado à função.

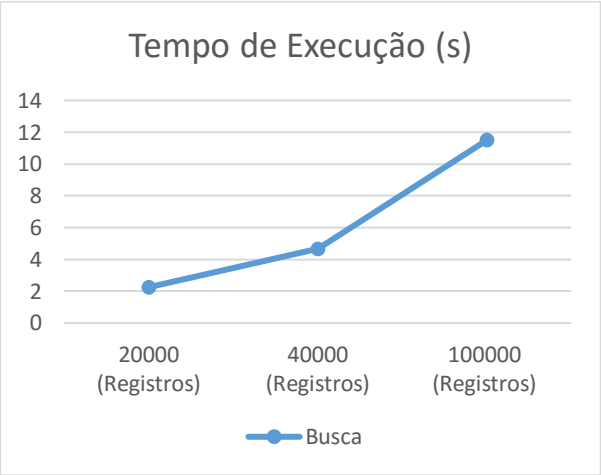

Figura 2 – Teste de tempo de execução no método de busca.

CONCLUSÕES

Foi apresentada uma implementação em linguagem C de uma estrutura de dados não bloqueante baseada em vetores de nós da fila. O algoritmo foi submetido a testes com alto índice de dados, a fim de se verificar seu desempenho, complexidade e análise de sintaxe. Observa-se que a função para exibir os dados custa mais para o hardware do que para a função que insere, pelo fato de terem instruções mais custosas ao hardware. Considerando a complexidade do código, testes mostraram um resultado satisfatório, indicando fácil entendimento e manutenção do algoritmo. O algoritmo tem potencial para novas pesquisas, incluindo realização de testes de desempenho de hardware e o aprimoramento do algoritmo, com a possibilidade da utilização da técnica não bloqueante em outras estruturas de dados.

AGRADECIMENTOS

Os autores agradecem ao programa PIBIFSP – IFSP – Edital 050/2015 – pelo auxílio financeiro a este trabalho de pesquisa, e a infraestrutura do IFSP Boituva pelo apoio no desenvolvimento deste trabalho.

REFERÊNCIAS

ARPACI-DUSSEAU, R.; ARPACI-DUSSEAU, A. C. Operating Systems: Three Easy Pieces. Edição Digital, Amazon Digital Services LLC, 2015.

Brooks, A. Static Analysis: Total Lines of Code / Source Lines of Code / Effective Lines of Code. University of Akureyri, Notas de aula 2008-2009, Disponível em: <http://staff.unak.is/andy/StaticAnalysis0809/metrics/loc. html>. Acesso em: 15 jul. 2016.

EVEQUOZ, C. Non-Blocking Concurrent FIFO Queues with Single Word Synchronization Primitives, International Conference on Parallel Processing, 2008, Disponível em: <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4625874&url=http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4625874>. Acesso em: 20 jul. 2016.

QUICOLI, P, Análise de código: métricas de qualidade e estimativas. 2016, Clube Delphi Magazine 147, DevMedia, Disponível em: <http://www.devmedia.com.br/analise-de-codigo-metricas-de-qualidade-e-estimativas-clube-delphi-magazine-147/26882>. Acesso em: 15 jul. 2016.

NAVAUX, Philippe O. A. **Introdução ao processamento paralelo,** Laboratório de Banco de Dados, Departamento de Ciência da Computação – UFMG, 1988, Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sbac-pad/1988/003.pdf>. Acesso em: 28 ago. 2016.

WATSON, A. H.; MCCABE, T. J. Structured Testing: A Testing Methodology, McCabe Software, 1996, Disponível em: <http://www.mccabe.com/pdf/mccabe-nist235r.pdf>. Acesso em: 25 jul. 2016.