PROGRAMAÇÃO PARALELA UTILIZANDO ESTRUTURAS DE DADOS NÃO BLOQUEANTE

GUILHERME PORTO BRITTO COUSIN[1]; ALAN S. DE ARAUJO[2]; GERSON GERALDO H. CAVALHEIRO[3]

[1]Universidade Federal de Pelotas – gpbcousin@inf.ufpel.edu.br
[2]Universidade Federal de Pelotas– asdaraujo@inf.ufpel.edu.br
[3]Universidade Federal de Pelotas– gerson.cavalheiro@inf.ufpel.edu.br

1. INTRODUÇÃO

Programas paralelos se beneficiam do hardware paralelo quando são capazes de explorar todos os processadores disponíveis de forma efetiva durante sua execução. A ociosidade de um processador, durante a execução do programa, representa perda potencial de desempenho (FRASER et al, 2007).

Este trabalho apresenta o projeto e a construção de uma estrutura de dados não bloqueante para o suporte à implementação dos serviços de um núcleo de escalonamento para um ambiente de execução *multithread*. Esta estrutura não bloqueante é definida por uma lista que gerencia o acesso de múltiplos *threads* a seu conteúdo. Os mecanismos implementados para este gerenciamento elimina o risco de um número massivo de *threads* corromper a estrutura de dados utilizada enquanto promove a execução eficiente da operações que a manipulam (inserções e remoções de elementos) pela eliminação do uso de mecanismos clássicos de sincronização baseados em primitivas de exclusão mútua. Os resultados apresentados neste trabalho mostram que o uso desta lista alcança um ganho de desempenho em suas operação de inserção e remoção em relação a soluções, baseadas em mecanismos bloqueantes, uma versão utilizando a STL, outra em código otimizado especialmente desenvolvido (SmartHeap).

A diminuição do sobrecusto de execução foi obtido pela eliminação do gargalo de acesso à lista quando é utilizado um bloqueio global da estrutura para garantir acesso em regime de exclusão mútua. Este trabalho diferencia-se por buscar uma solução eficiente de listas não bloqueantes adaptadas aos mecanismo de escalonamento do ambiente Anahy3.

Como resultado, mostramos que a introdução de listas não bloqueantes no núcleo de escalonamento de Anahy3 pode gerar ganhos de desempenho, uma vez que estas listas permitem reduzir a contenção da execução dos *threads* que dependem de mecanismos de bloqueio para o acesso a lista.

É importante dizer que estes algoritmos não são genéricos e usualmente desenvolvidos para um determinado fim. No nosso caso o fim é o núcleo de escalonamento.

2. METODOLOGIA

As estruturas de dados são um modo particular de armazenamento e organização de dados em um computador de modo que possam ser usados de modo eficiente. Estruturas de dados e algoritmos são temas fundamentais, sendo utilizados nas mais diversas áreas do conhecimento e com os mais diferentes propósitos de aplicação. Sabe-se que algoritmos manipulam dados e que estes dados devem ser mantidos de forma organizada em uma estrutura de dados no interior de um programa. A organização e os métodos para manipular essa estrutura é que lhe conferem singularidade.

[1]Bolsista PBIC CNPq
"FAPERGS/PqG (11/1065-1), PRONEX/FAPERGS/CNPq (10/0042-8)"

No caso em que o programa é concorrente, quando as estruturas de dados são compartilhadas entre diferentes fluxos de execução, é necessário controlar a concorrência das operações realizadas sobre elas. Caso não seja controlada esta concorrência pode haver um problema de inconsistência no acesso aos dados (SPOONHOWER et al, 2009).

Os algoritmos não-bloqueantes baseiam-se no princípio de que conflitos de sincronização são raros e devem ser tratados como exceção (VALOIS, 1995). Quando um conflito é detectado em uma estrutura, a operação que detectou o conflito é abortada e sua execução é refeita. Algoritmos não-bloqueantes de sincronização evitam *deadlocks* e inversões de prioridade. Estes algoritmos devem ser, quando implementados, tolerantes a falhas de *threads*. Como consequência o programa em execução não sofrerá degradação de desempenho por preempção, forçada pela chamada de uma operação de sincronização executada em nível sistema.

O algoritmo proposto foi implementado como uma lista duplamente encadeada. A estrutura concebida, possui dois nodos fixos, ou seja, nodos que não podem ser removidos. Entre estes nodos fixos, os elementos da lista, os quais contêm informações relevantes ao programa em execução são inseridos e removidos. Desta forma, há a possibilidade de inserção e de remoção tanto no início quanto no final da lista sem perda da consistência da estrutura da lista. Para manter a consistência da estrutura, é utilizada uma *flag*, que identifica se o nodo em questão já esta sendo utilizados por outros *threads*.

A inserção é feita do seguinte modo: o *thread* tenta alterar a *flag* do nodo principal da extremidade, caso haja sucesso, o *thread* tenta alterar a *flag* do nodo seguinte caso consiga alterar esta *flag*, então o elemento é inserido entre os dois nodos, e depois as *flags* alteradas voltam ao estado original. Caso alguma das *flags* já tenham sido alteradas por outro *thread*, o algoritmo libera as *flags* alteradas por ela e retorna ao início da inserção, evitando assim *deadlocks* e assumindo que a inserção não foi realizada.

Na remoção, o *thread* tenta alterar a *flag* do nodo principal da extremidade na qual deseja-se fazer a remoção, logo a pós efetuar a troca deste nodo, o *thread* tenta alterar as *flags* dos outros dois nodos seguinte. Caso haja sucesso, o elemento mais próximo ao nodo principal é removido. Caso alguma das *flags* já tenham sido alterada por outro *thread*, o algoritmo libera as *flags* já alteradas por ele, e volta ao início da remoção assumindo que a remoção não foi realizada e nova tentativa deve ser feita.

3. RESULTADOS E DISCUSSÃO

Nesta seção apresentamos uma avaliação do desempenho de uma implementação do algoritmo não bloqueante apresentado. Os resultados apresentados foram executados em computador, em uso dedicado, dotado com um processador Intel Core i7-2600 CPU @ 3.40GHz x 8, com memória de 8GB. Os tempos referem-se a uma média de 100 execuções, não tendo sido observado desvio padrão acima de 3%.

Os estudos de caso realizados também foram submetidos à execução sobre uma implementação convencional de listas utilizando a STL e a *Smart Heap* atualmente em uso em Anahy(ARAUJO, 2013). A *Smart Heap* e tem como principal característica a otimização da manipulação da memória dinâmica. A STL, por sua vez, consiste em uma biblioteca muito popular de estruturas de dados, no entanto ela é *thread-safe* apenas *threads* no caso de diferentes *threads* acessarem simultaneamente diferentes containers. No caso onde um único

container é objeto de acessos concorrentes, mecanismos de sincronização externos devem ser utilizados. Os estudos de caso realizados correspondem a inserções e remoções de itens na estrutura de dados intercalados pela execução de uma carga sintética correspondente ao cálculo recursivo da série de Fibonacci para a posição 20. As figuras 1.a) e 2.a) correspondem, respectivamente, aos tempos obtidos na inserção e remoção de 10.000 elementos. As figuras 1.b) e 2.b) apresentam os tempos de inserção e remoção de 100.000 elementos. As figuras 1.c) e 2.c) apresentam os tempos de inserção e remoção de 1.000.000 elementos.

E possível observar nos resultados, que com o maior número de elementos inseridos na lista, quando da execução com mais de um *thread,* a tendência e que o desempenho melhore em relação às outras abordagens comparadas. A medida que aumenta o número de remoções, nesta mesma situação, o desempenho também evolui, mesmo possuindo um mecanismo de tratamento de conflitos. No estudo de caso realizado, onde os *threads* apenas inserem ou removem itens na lista, observou-se que o melhor desempenho foi obtido com oito *threads*, o que é compreensível pelo fato dos *threads* poderem manipular extremidades distintas da lista com menor probabilidade se conflito. Outro aspecto a ser considerado, quando da análise destes resultados, e que está sendo verificado o desempenho das estratégias quando não inseridas no núcleo de escalonamento de Anahy.

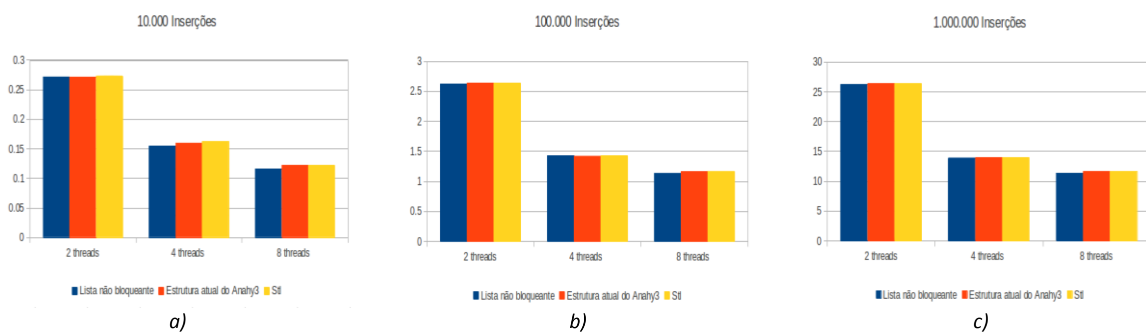

Figura 1. Comparação dos tempos de inserção em diferentes estratégias.

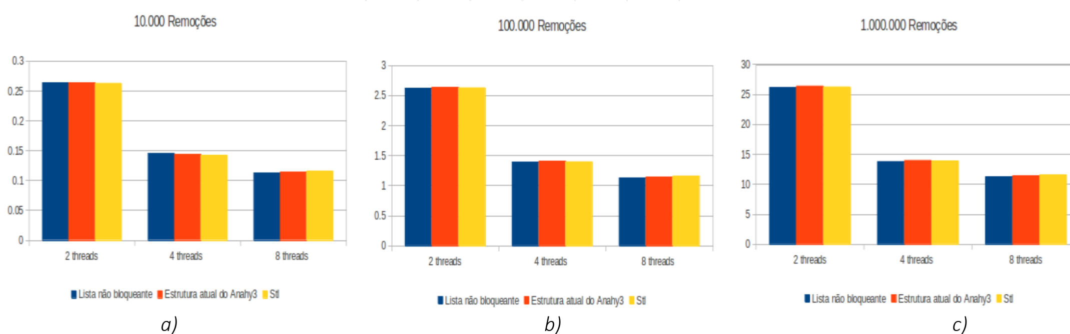

Figura 2. Comparação dos tempos de remoção em diferentes estratégias.

É de se esperar um desempenho satisfatório quando do uso efetivo no núcleo de execução de Anahy. Este núcleo e composto por processadores virtuais (PVs) que consistem em *threads* sistema responsáveis por executar os *threads* do usuário. Na concepção deste ambiente, cada PV gerência, em uma lista local, os *threads* usuário criados localmente e prioriza a execução destes *threads* locais em detrimento de *threads* criados em outros PVs. Caso a lista de um PV fique

vazia, uma operação global de escalonamento transfere um trabalho de um PV, escolhido segundo alguma estratégia de escalonamento implementada em Anahy, para esta lista, evitando perda de capacidade de processamento. O custo computacional inserido, para computação da 20 posição de Fibonacci tem por função simular o comportamento de uma aplicação em execução no ambiente Anahy.

4. CONCLUSÕES

As estruturas não bloqueante são importantes, pois reduzem o acesso a seção crítica, tendo assim uma maior paralelização das operações de inserção e remoção.

Conforme observamos na Seção **3**, quando o número de *threads* é muito alto, esta estrutura não possui um ganho potencial, pois o número de conflitos dos *threads* é bastante grande. Podemos avaliar também que conforme o número de inserções e aumenta, o ganho em diferença as outras estruturas também aumenta. Em contrapartida, quando o número de conflitos potenciais é pequeno, observa-se que o desempenho da estratégia proposta é bastante satisfatório.

Desta forma entendemos que a estratégia proposta possa a vir ser integrada com sucesso em Anahy, uma vez que o número de interações entre os PVs pode ser limitada conforme a estratégia de escalonamento utilizada.

5. REFERÊNCIAS BIBLIOGRÁFICAS

FOMITCHEV M. and RUPPERT E., “Lock-free linked lists and skip lists,” in **Proceedings of the twenty-third annual ACM Symp. on Principles of distributed computing**. PODC ’04. New York, NY, USA: ACM, 2004, pp. 50–59.

HENDLER N. S. D. and YERUHALMI L., “A scalable lock-free stack algorithm,” in **Proc. of the 16th ACM Symp. On Parallelism in Algorithms and Architectures**, 2004, pp. 206–215.

FRASER K. and HARRIS T., “Concurrent programming without locks,” **ACM Transactions on Computer Systems**, vol. 25, no. n.2, 2007.

VALOIS J. D., “Lock-free linked list using compare-and-swap,” in In **Porceding of the 14th Annual ACM Symp. On Principles of Distributed Computind**, 1995, pp. 214 – 222.

SPOONHOWER D., BLELLOCH G. E., GIBBONS P. B., and HARPER R., “Beyond nested parallelism: tight bounds on workstealing overheads for parallel futures,” in **Proceedings of the twenty-first annual Symp. on Parallelism in algorithms and architectures**. ACM, 2009, pp. 91–100.

ARAUJO, A. S., **“Anahy-3: Um novo ambiente de execução otimizado para arquiteturas multicore**,”.2013. Monografia (Bacharelado em Ciência da Computação) – Universidade Federal de Pelotas

KUNZ L., “**Memórias transacional em hardware para sistemas embarcados multiprocessados conectados por redes em chip**,”.2010 Master’s thesis, URGS, Porto Alegre,