COMPILADOR DIRIGIDO POR SINTAXE QUE TRADUZ LINGUAGEM CMTJAVA PARA EXPRESSÕES LAMBDA DO JAVA 8

ÁLVARO GONÇALVES[1]; ANDRÉ RAUBER DU BOIS[2]; JERONIMO RAMOS[3]; MAURÍCIO LIMA PILLA[4];

[1]Universidade Federal de Pelotas (UFPel) – ajgdsantos@inf.ufpel.edu.br
[2]Universidade Federal de Pelotas (UFPel) – dubois@inf.ufpel.edu.br
[3]Universidade Federal de Pelotas (UFPel) – jdcramos@inf.ufpel.edu.br
[4]Universidade Federal de Pelotas (UFPel) – pilla@inf.ufpel.edu.br

1. INTRODUÇÃO

A evolução dos processadores por muito tempo foi baseada na frequência em que o hardware trabalhava, dessa forma, aplicações sequenciais sempre foram ganhando desempenho enquanto a frequência em que o processador trabalhava aumentava. Contudo, em dado momento, alguns fatores tecnológicos e arquiteturais não permitiram que se aumentasse mais a frequência.

No lado tecnológico, problemas como superaquecimento e dissipação de potência inviabilizaram aumentar a frequência de trabalho dos processadores. Assim a industria viu-se forçada a adotar um novo modelo arquitetural para continuar a aumentar o desempenho do hardware, dessa forma, os esforços foram voltados para processadores *multicore*.

Esses novos processadores permitiram um ganho de desempenho considerável na execução de programas. O programa era divido em partes discretas que eram resolvidas concorrentemente ou paralelamente. Comparando esta abordagem aos programas sequenciais, notou-se um grande aumento de desempenho (HASSELBRING, 2000). Entretanto, houve um aumento na complexidade para a sua programação devido aos mecanismos de bloqueio(*locks*) necessários para o controle.

Nesse contexto, surge um novo modelo para abstrair e controlar a concorrência, memórias transacionais (HERLIHY; MOSS, 1993). Nessa abordagem, todo acesso a memórias compartilhadas é realizado por meio de transações. Para o programador o controle de seções críticas, antes feito com mecanismos de bloqueio, nos quais eram de total responsabilidade da habilidade do programador, agora viram apenas explicitações das seções críticas do programa. O sistema transacional é encarregado em como executar aquela seção crítica de forma atômica, abstraindo toda complexidade para o programador.

Em CMTJava (ECHEVARRIA; DU BOIS, 2010) o modelo de memórias transacionais é implementado em software. CMTJava é uma extensão de Java para programação utilizando o modelo de memórias transacionais, que baseia-se na ideia de STM Haskell (HARRIS et al., 2005). A linguagem CMTJava é atualmente implementada em três camadas: a própria linguagem CMTJava na camada de alto nível, uma linguagem intermediária e um sistema de baixo nível que dá suporte as transações.

Por sua camada de alto nível adotar abstrações não suportadas em Java, é necessário traduzi-la para uma linguagem intermediária. Em sua implementação atual (LEÃO BANDEIRA, 2011) essa linguagem intermediária utiliza uma proposta antiga de closures para Java, o BGGA closures (GAFTER, 2015), pois essas não eram suportadas nativamente. Este trabalho tem como objetivo a implementação de um compilador para transformar código CMTJava em *bytecodes*, para isso, é utilizado as novas tecnlogias da linguagem Java,

expressões lambda (ORACLE, 2015). *Closures* agora são suportados nativamente pela linguagem Java 8 por meio de expressões lambda, porém, com sintaxe e semântica diferentes das utilizadas na solução atual.

2. METODOLOGIA

Para facilitar o entendimento do projeto da CMTJava, suas camadas são apresentadas na Figura 1. Atualmente CMTjava é dividida em três camadas com funcionalidades distintas, que rodam em cima da Máquina Virtual Java (JVM), a camada em vermelho representa a área onde esse trabalho propõe a mudança.

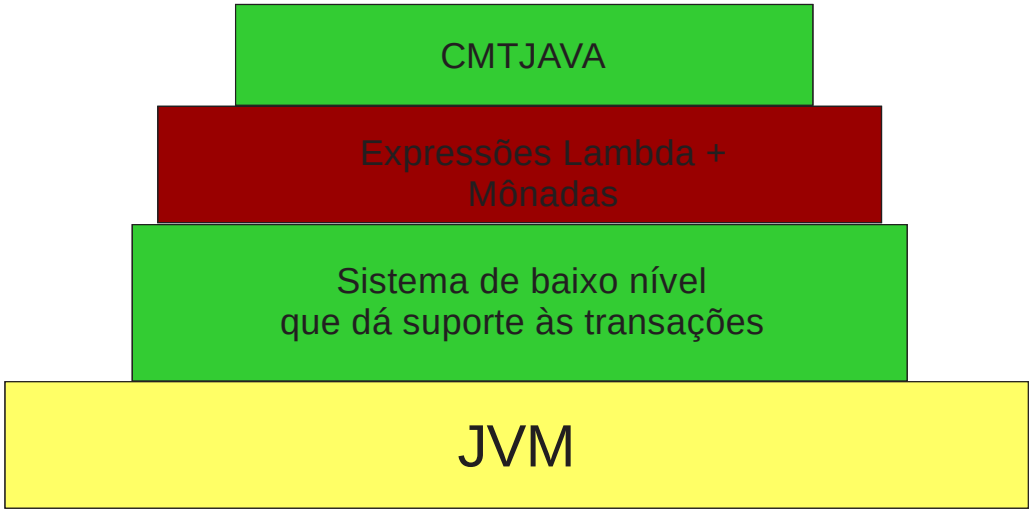

Figura 1. Modelo de camadas da CMTJava

A **Camada da Linguagem CMTJava** é uma extensão da linguagem Java que provém os beneficios de memórias transacionais. CMTJava permite a programação utilizando todos benefícios de memórias transacionais em uma linguagem orientada a objetos. Sem alterar drasticamente as caracteristicas da linguagem, CMTJava oferece toda uma interface de objetos transacionais e construções que trazem uma maior abstração para a programação

A **Camada de Expressões Lambda + Mônadas**, Na segunda camada do CMTJava, representada em vermelho na Figura 1, é a linguagem intermediária. Essa camada é responsável pela tradução das abstrações da camada CMTJava para Java puro. A implementação atual (LEÃO BANDEIRA, 2011) é feita com uma abordagem antiga de *closures*, o BGGA (GAFTER, 2015), e Mônadas. Este trabalho propõe uma nova abordagem nessa camada, utilizando em sua linguagem intermediária Expressões Lambda (*closures*), nativas do novo Java 8, juntamente com mônadas, para representar as abstrações dos objetos transacionais.

A **Camada de Baixo Nível** é responsável pelo suporte as transações. Esse sistema faz todo controle de detecção de conflitos e versionamento de dados em áreas críticas do programa.

A Figura 2 representa uma visão geral do processo de compilação para um programa CMTJava. Mais detalhadamente, a tradução compreende a parte de transformação de código. Nesta etapa o compilador se encarrega de traduzir o código CMTJava para código Java puro, além disso, também é feita geração das interfaces das classes que implementam os objetos transacionais. Por fim a parte

compreendida por *síntese*. Nessa fase nosso programa CMTJava já foi traduzido para um *.java*. Na *síntese* já é possível trabalhar com o compilador primário do java, o *javac*, para gerar os *bytecodes*.

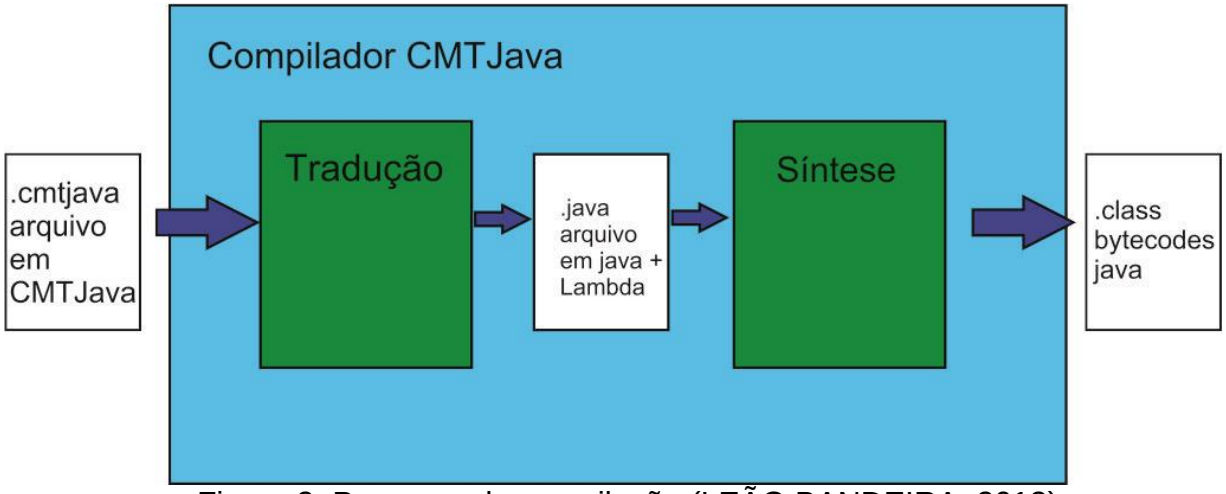

Figura 2. Processo de compilação (LEÃO BANDEIRA, 2013)

3. RESULTADOS E DISCUSSÃO

Até o momento, os esforços deste trabalho foram destinados a uma revisão bibliográfica que engloba, principalmente, conceitos e tecnologias. Primeiramente, foi realizado um estudo sobre programação paralela, onde foram verificadas as justificativas para seu uso e entendimento de suas principais técnicas. A partir disso, o estudo foi voltado para memórias transacionais, que apresentou-se ser uma ótima solução para tratar as dificuldades encontradas com a programação de máquinas multicore. Foi verificada a existencia de diversos modelos que implementam memória transacional em software. assim como a CMTJava.

A CMTJava é uma linguagem de alto nível que suporta memórias transacionais, além disso, CMTJava trás abstrações e atômicidade nas operações executadas paralelamente. Para o entendimento da antiga proposta de compilador (LEÃO BANDEIRA, 2011) foram revistos conceitos necessários para construção de compiladores, assim como ferramentas para o auxílio dessa construção, o ANTLR (PARR, 2007).

Na implementação atual da CMTJava, é utilizada uma abordagem antiga de closures, internamente à linguagem, com o objetivo de trazer maior abstração ao programador. Porém, Java 8 traz como uma de suas novas tecnologias, expressões lambda, agora nativas. Na continuidade do trabalho, estas irão substituir a implementação antiga de closures. Será realizada uma comparação de desempenho entre os programas compilados com ambas as versões do compilador da linguagem e, espera-se, que os programas da nova versão apresentem um maior desempenho.

Para auxiliar na realização das alterações no compilador da linguagem será utilizada a ferramenta ANTLR, como já é feito atualmente. Finalmente, acredita-se que não serão necessárias mudanças nas camadas da linguagem CMTJava e do sistema de baixo nível que dá suporte às transações.

4. CONCLUSÕES

Este trabalho apresentou a proposta de um novo compilador para o CMTJava. A implementação existente, como dito anteriormente, está estagnada com tecnologias antigas, assim, busca-se dar continuidade ao projeto CMTJava. Até o momento o foco foi dado na revisão bibliográfica. Ao decorrer das próximas etapas pretende-se realizar a construção do compilador. Por fim, com a nova implementação do compilador, será possível realizar a comparação entre programas compilados em ambas as versões da linguagem e, assim, espera-se que o novo compilador traga um ganho significativo de desempenho.

5. REFERÊNCIAS BIBLIOGRÁFICAS

HERLIHY, M.; MOSS, J. E. B. **Transactional memory: Architectural support for lock-free data structures.** [S.l.]: ACM, 1993. v.21, n.2.

ECHEVARRIA, M. G.; DU BOIS, A. R. **Uma Linguagem de Domınio Especıfico para Programação de Memorias Transacionais em Java.** [S.l.], 2010.

HARRIS, T.; MARLOW, S.; PEYTON, S.; HERLIHY, J. M. **Composable memory transactions.** In: 2005. Anais… ACM Press, 2005. P.48–60.

GAFTER, N. **Closures for the Java Programming Language.** Disponivel em<http://www.javac.info/>, Acesso em 10 de jun. 2015.

LEÃO BANDEIRA, R. **Compilador para a linguagem CMTJava.** Pelotas/RS:[s.n.], 2011.

ORACLE. **Lambda Expressions**. Disponivel em <https://docs.oracle.com/ javase/tutorial/java/javaOO/lambdaexpressions.html>, Acesso em 20 de jun.2015.

PARR, T. **The definitive ANTLR reference: building domain-specific languages.** [S.l.], 2007.

HASSELBRING, W. **Programming Languages and Systems for Prototyping Concurrent Applications.** ACM COMPUTING SURVEYS, [S.l.], v.32, 2000.