NOPL-Erlang: Programação multicore transparente em linguagem de alto nível

Fabio Negrini[1], Adriano Francisco Ronszcka[1], Robson Ribeiro Linhares[1], João Alberto Fabro[1], Paulo Cézar Stadzisz[1], Jean Marcelo Simão[1]

[1] Prog. de Pós-Graduação em Engenharia Elétrica e Informática Industrial Universidade Tecnológica Federal do Paraná (CPGEI-UTFPR)
80230-901 – Curitiba – PR – Brasil

{negrini,ronszcka}@alunos.utfpr.edu.br,
linhares,fabro,stadzisz,jeansimao}@utfpr.edu.br

Abstract. *The growth of the computational capacity has been reached by means of the increment of cores inside a same processor. This solution, however, greatly increases the complexity of programming, forcing developers to make use of programming concurrency techniques. In this paper the NOPL is presented, a programing language for NOP with decoupling and non-sequential properties. NOPL is a high level multicore transparent programming language. This paper presents the NOPL language and its integration with Erlang architecture to explore concurrent execution. In the results it is possible to verify the expressive reduction in the execution time as the number of cores is increased.*

Resumo. *O aumento da capacidade computacional dos processadores tem sido alcançado por meio do incremento de núcleos dentro de um mesmo processador. Esta solução, entretanto, aumenta consideravelmente a complexidade de programação, obrigando desenvolvedores fazerem uso técnica de programação concorrente. Neste artigo é apresentado a NOPL, linguagem própria do PON com propriedades desacoplante e não sequencial. NOPL é uma linguagem que permite a programação concorrente de maneira transparente e em alto nível. O artigo apresenta a linguagem NOPL e sua integração com a arquitetura Erlang para explorar execução concorrente. Nos resultados é possível verificar a redução significativa no tempo de execução à medida em que se aumenta o número de núcleos disponíveis.*

1. Introdução

O fato de os semicondutores estarem próximos do limite de velocidade tem instigado pesquisas ao redor do mundo para encontrar formas alternativas de aumentar o desempenho computacional [DeBenedictis 2017]. Neste sentido, o aumento da densidade de integração tem sido aproveitado pelos fabricantes para a implementação de múltiplos núcleos em uma mesma pastilha, o que é usualmente chamado de multicore. Embora, em tese, o aumento do número de unidades de processamento em paralelo permita aumentar o desempenho de execução da computação, na prática isto depende de software que explore adequadamente o paralelismo.

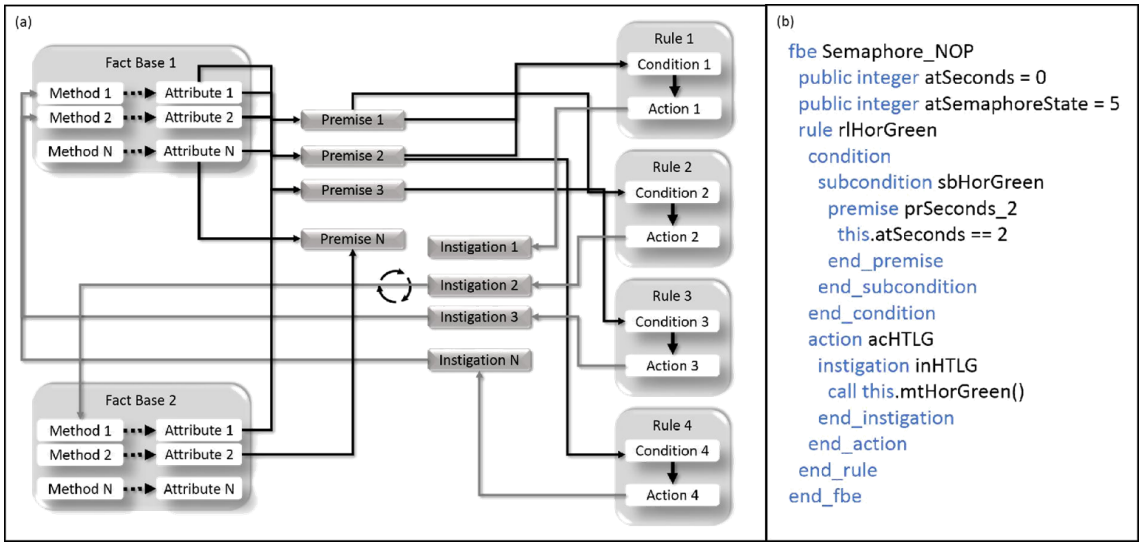

Figura 1. a) Colaboração por notificações em PON [Ronszcka et al. 2017]
b) Exemplo de código em NOPL

Entretanto, a construção de software de forma paralela e/ou adequada para execução em arquiteturas multicore impõe naturalmente dificuldades de abstração aos desenvolvedores em função justamente da nova dinâmica de execução paralela [Borkar and Chien 2011].

Neste contexto, este artigo explora uma abordagem alternativa de desenvolvimento de software: o Paradigma Orientado a Notificações (PON). O PON é, em suma, um paradigma emergente para o desenvolvimento de sistemas computacionais com maior nível de abstração, o que facilita a construção de programas que explorem adequadamente o paralelismo/distribuição em comparação com sistemas baseados em subparadigmas tradicionais, como Programação Procedimental, Programação Orientada a Objetos (POO) e Sistemas baseados em Regras (SBR) [Simão and Stadzisz 2009].

Neste artigo são apresentados avanços alcançados com a evolução NOPL (Notification Oriented Programming Language) [Ronszcka 2019] e seu compilador próprio, criando um novo target baseado na tecnologia Erlang. Este novo target de compilação é suportado por um framework sobre uma máquina virtual Erlang. A combinação da NOPL com este compilador específico proporciona a característica de programação em alto nível e execução concorrente de forma transparente.

2. Revisão da Literatura

2.1. PON e NOPL

O PON é um paradigma de programação emergente que apresenta similaridades a sistemas baseados em regras. Estruturalmente, entretanto, o software PON é representado na forma de entidades, nomeadamente as entidades chamadas elementos da Base de Fatos (FBE – Fact Base Element) e as Regras (Rules). As entidades FBE são utilizadas para representar objetos do mundo (real ou abstrato) em um sistema computacional, por meio de estados (atributos) e serviços (métodos). Os métodos PON, entretanto, são bastante pontuais, podendo apenas realizar operações matemáticas ou ações diretas, sem possuir lógica intrínseca de tomada de decisão (não possuem nem comandos de seleção (if) nem estruturas de repetição (for/while). As entidades Rules, por sua vez, definem o cálculo lógico-causal a ser efetuado sobre os estados dos FBEs, controlando a execução de suas ações. A colaboração entre estes elementos ocorre por meio de notificações diretas das

entidades que constituem cada FBE (i.e., Attributes e Methods) e Rules (i.e., Premises, Conditions, Actions e Instigations) [Simão and Stadzisz 2009]. O diagrama apresentado na Figura 1a apresenta graficamente os relacionamentos entre as entidades citadas. Neste contexto é proposta a NOPL - a linguagem de programação baseada no paradigma PON que traz consigo os princípios fundamentais do PON: a) facilidade de desenvolvimento de software em alto nível; b) isenção de redundâncias estruturais, e c) desacoplamento explícito dos elementos. A tecnologia de compilação da NOPL consiste em um tradutor que se utiliza de um meta-modelo na forma de um grafo-framework permitindo sua derivação para uso em plataformas distintas [Ronszcka 2019]. A figura 1b apresenta um exemplo de código em NOPL.

2.2. Erlang

A linguagem Erlang surgiu na década de 1980 nos laboratórios de Ciência da Computação da Ericsson para suportar aplicações distribuídas e tolerantes a falhas. Em vez de fornecer processos que compartilham memória, cada processo Erlang é executado em seu próprio espaço de memória e possui seu próprio heap e pilha evitando interferências indevidas entre os processos. Erlang aplica a abordagem de concorrência Modelo de Ator de forma que os processos se comunicam entre si via passagem de mensagens assíncronas [Cesarini and Thomson 2009]. Contudo, apesar das inúmeras facilidades, a granularidade dos atores ainda depende da habilidade do desenvolvedor em pensar e construir módulos suficientemente desacoplados.

3. Materiais e métodos

3.1. PON e o modelo de atores

O modelo de atores de Erlang [Hewitt et al. 1973] requer algum meio para prover atores com granularidade apropriada para melhor aproveitar o balanceamento de processamento dos atores em multicore. Neste sentido, as tecnologias PON e Erlang poderiam ser conjugadas. A proposta deste artigo é o aproveitamento da arquitetura concorrente Erlang com aderência dos elementos do paradigma PON em processos com estados que se comunicam por meio de mensagens. Após uma análise minuciosa de cada elemento PON, foi possível chegar a uma modelagem de alto nível de cada um de seus elementos em forma de microatores. Com esta modelagem, cada elemento apresentado na Figura 1a é traduzido em um processo concorrente Erlang e o mecanismo de notificações, por sua vez, traduzido em mensagens assíncronas.

3.2. Framework NOP Erlang/Elixir

Uma vez feita a análise de aderência dos elementos do NOP para elementos com comportamento de atores, avançou-se a para a codificação. Optou-se pela utilização da linguagem Elixir, pois esta é totalmente imersa na plataforma Erlang e ainda disponibiliza mecanismos essenciais para a programação estruturada [Thomas 2018]. Estes mecanismos resultantes podem ser utilizados de maneira independente e estão disponíveis para a comunidade Elixir para testes e avaliações. O resultado desta codificação é apresentado em forma de pacote HEX para facilitar sua distribuição e utilização. Em suma, criou-se um framework NOP Erlang/Elixir que opera sobre a lógica do PON e não sobre a tradicional lógica funcional-imperativa sobre a qual normalmente os sistemas são programados nesta plataforma.

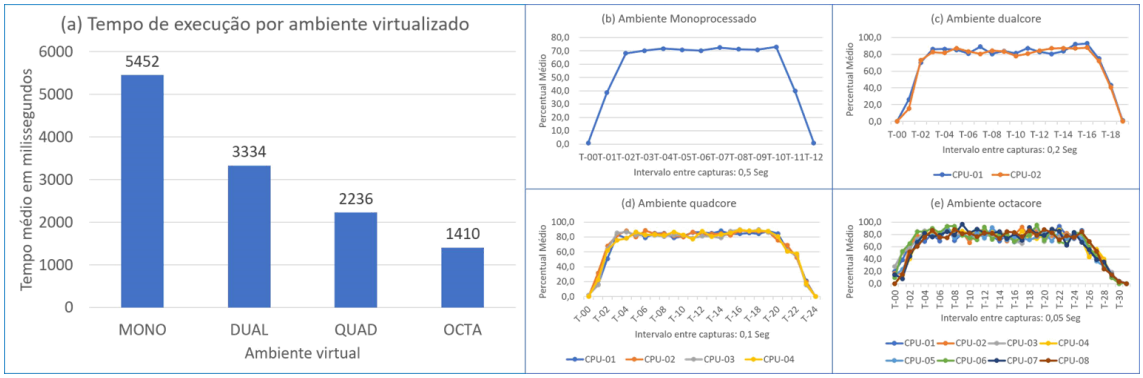

Figura 2. Resultados das simulações em ambientes EC2 tipo T2 Amazon AWS

3.3. Target Framework NOP Erlang/Elixir

Por fim, seguiu-se a etapa de construção do compilador NOPL especializado para o target framework NOP Erlang/Elixir. Esta etapa é baseada no método MCPON (Método de Compilação PON) que permite a construção de compiladores por meio de um grafo de entidades PON e um conjunto de diretrizes que facilitam a integração da linguagem NOPL com o novo target gerado [Ronszcka 2019]. Isto posto, os elementos identificados no grafo-framework são então convertidos em elementos do framework e suas ligações devidamente alinhadas. O resultado é uma solução que permite desenvolver aplicações em NOPL e automaticamente compilá-las para microatores em PON sobre a plataforma Erlang. Assim, o desenvolvedor dispõe de recursos para um melhor aproveitamento da capacidade de concorrência através da arquitetura concorrente Erlang.

4. Resultados e discussões

4.1. Experimento CTA

Para avaliar a capacidade de processamento concorrente multicore com a solução proposta, nomeada de tecnologia NOPL-Erlang, propôs-se como experimento um programa em NOPL para controle de trânsito automatizado (CTA) definido em [Renaux et al. 2015]. O objetivo do CTA é alternar os estados de semáforos durante um ciclo de noventa segundos. Este ambiente foi reproduzido para um conjunto de dez quadras horizontais por dez quadras verticais totalizando cem semáforos. Depois, foram simulados ciclos de 2000 segundos em sequência para todos os semáforos.

Foi utilizado um conjunto de quatro ambientes virtualizados do tipo T2 Ubuntu Server 14.04 LTS 64 bits e armazenamento SSD disponibilizados pela Amazon. Os resultados podem ser vistos na Figura 2. A Figura 2a representa o tempo médio de execução (em milissegundos) em cada ambiente virtualizado. Como é possível perceber no cenário apresentado, há uma redução de 73,27% tempo de execução do programa de simulação quando comparado com o ambiente mono core com o octa core. As Figuras 2b, 2c, 2d e 2e demonstram uma distribuição balanceada entre os núcleos respectivamente para ambientes mono, dual, quad e octa core, sugerindo que a execução aproveitou o potencial disponível dos núcleos de forma balanceada em prol de uma otimização e redução do tempo total de execução.

5. Conclusão e trabalhos futuros

Como é possível verificar, com a tecnologia NOPL-Erlang, conforme o mesmo programa é executado em ambiente com múltiplos núcleos (multicore), o seu tempo de execução se reduz, ao passo que os núcleos permanecem todos com balanceamento de utilização durante todo o processamento, demonstrando que a execução está conseguindo se ocupar de todos os núcleos para este experimento, que avaliou ambientes de até oito núcleos. Todo este aproveitamento é alcançado com programação em alto nível e estruturada, sem interferência ou conhecimento do programador em ambientes multicore devido ao trabalho de união da linguagem NOPL e o desacoplamento implícito próprio do PON com a arquitetura concorrente Erlang, por meio de um framework que explora adequadamente as características do PON. Somente com Erlang, tal concorrência seria possível com o desenvolvimento de alguma técnica de programação paralela demandando conhecimento adicional. Como trabalhos futuros sugere-se avaliações mais profundas em relação à utilização com maior quantidade de núcleos para melhor avaliar o comportamento desta nova tecnologia. Outro trabalho futuro sugerido é o de aprimorar a tecnologia NOPL-Erlang para garantir determinismo conforme detalhado em [Simão and Stadzisz 2010], os quais não foram explorados nos experimentos apresentados neste artigo.

Referências

Borkar, S. and Chien, A. A. (2011). The future of microprocessors.

Cesarini, F. and Thomson, S. K. (2009). Erlang Programing. O'Reilly Media.

DeBenedictis, E. P. (2017). It's time to redefine moore's law again. Computer, 50(2):72–75.

Hewitt, C., Bishop, P., and Steiger, R. (1973). A universal modular actor formalism for artificial intelligence.

Renaux, D. P. B., Linhares, R. R., Simão, J. M., and Stadzisz, P. C. (2015). Cta conops. http://www.dainf.ct.utfpr.edu.br/~douglas/CTA_CONOPS.pdf.

Ronszcka, A. (2019). Método para a criação de linguagens de programação e compiladores para o paradigma orientado a notificações em plataformas distintas. Teste. Doutorado em Computação Aplicada - CPGEI. Universidade Tecnológica Federal do Paraná (UTFPR) Curitiba - PR.

Ronszcka, A., Valença, G., Linhares, R., Stadzisz, P., and Simão, J. (2017). Notification-oriented paradigm framework 2.0: An implementation based on design patterns. IEEE Latin America Transactions, 15(11).

Simão, J. M. and Stadzisz, P. C. (2009). Inference based on notifications: A holonic metamodel applied to control issues. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 39(1):238–250.

Simão, J. M. and Stadzisz, P. C. (2010). Mecanismo de resolução de conflito e garantia de determinismo para o paradigma orientado a notificações (pon). Pedido de patente nro PI1000296-0. Data de depósito no INPI: 02/2010.

Thomas, D. (2018). Programing Elixir. The Pragmatic Bookshelf.