LingPON: Programação multicore transparente já é uma realidade

Fabio Negrini
CPGEI / UTFPR
Avenida Sete de Setembro, 3165
Curitiba-PR - CEP 80.230-910
E-mail: negrini@alunos.utfpr.edu.br

***Resumo*—A busca por aumento de capacidade computacional dos processadores, hoje próxima do seu limiar, tem sido suplantada com aumento do número de núcleos dentro de um mesmo processador. Se por um lado, esta nova abordagem permite ampliar exponencialmente a capacidade computacional, por outro lado ela aumenta consideravelmente a complexidade do uso destes recursos obrigando programadores e desenvolvedores a pensarem em programação paralela. Fato este que traz problemas não existentes na programação sequencial como: gerenciamento de fluxo, acesso a recursos globais e sincronização de eventos, a fim de garantir o determinismo do processo. Estas dificuldades se dão em parte pelo forte acoplamento das linguagens de programação e por serem prioritariamente pensadas em execução sequencial. Neste artigo é apresentado o LingPON como uma proposta de programação alto nível transparente multicore, ou seja, uma linguagem que promove o uso otimizado dos processadores sem que o programador precise se preocupar com qualquer tipo de ferramenta ou técnica de paralelismo. Em materiais e métodos é apresentada a arquitetura proposta LingPON como linguagem desacoplada e não sequencial, bem como é feita integração desta com a arquitetura Erlang para alcançar concorrência. Nos resultados é possível verificar a redução expressiva no tempo de execução de um programa LingPON à medida em que se aumenta o número de núcleos disponíveis sugerindo o uso concorrente destes núcleos de maneira transparente.**

I. Introdução

O fato de estarmos próximos do limite da Lei de Moore [1] têm instigado pesquisas ao redor do mundo para encontrar formas alternativas de aumentar desempenho computacional. 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. Este, por sua vez, depende de técnicas sequenciais de desenvolvimento e particularmente de execução, as quais geralmente e naturalmente impõem dificuldades de abstração aos desenvolvedores em função justamente da dinâmica de execução paralela [2] [3] [4]. Esta programação paralela traz um grau muito maior de dificuldade em relação à programação sequencial e isto se dá majoritariamente pelos seguintes pontos: 1.Gerenciamento do fluxo de controle ao longo de vários encadeamentos de execução;2. Gerenciamento do acesso a dados globais, e;3.Sincronização eventos de tal forma que a execução do programa seja determinística e velocidade paralela seja alcançada [5].

Neste ambiente surge uma técnica alternativa de desenvolvimento de software: Paradigma Orientado a Notificações (PON). O PON é, em suma, uma nova abordagem para o desenvolvimento de sistemas computacionais. O PON tende a apresentar melhor desempenho, maior nível de abstração e proporciona facilidades para o desacoplamento de entidades, o que facilita paralelismo/distribuição, em comparação com sistemas baseados em subparadigmas tradicionais. Dentre esses salientam-se a Programação Procedimental e a Programação Orientada a Objetos (POO) [6] do Paradigma Imperativo (PI), assim como os Sistemas baseados em Regras (SBR) do paradigma declarativo (PD).

Neste artigo são apresentados avanços alcançados com a LingPON [7] - a linguagem de programação baseada no PON - que, suportada por uma máquina virtual Erlang, permite programação alto nível, compilação e execução concorrente. Esta concorrência se dá de forma natural em função das características de desacoplamento da linguagem que, aliada à arquitetura multicore da máquina virtual BEAM disponibilizada pela arquitetura Erlang/OTP [8], permitem o uso de seus métodos de balanceamento. Nos materiais e métodos são apresentados os detalhes desenvolvimento que permitem a união do LingPON [9] com a arquitetura Erlang.

Nos resultados é possível verificar que, conforme aumenta-se o número de núcleos, o mesmo programa reduz seu tempo de execução ao passo que mantêm todos os núcleos com equilíbrio de ocupação, sugerindo uma distribuição transparente dentro da arquitetura proposta.

II. Revisão da literatura

A. LingPON

O LingPON é a linguagem de programação baseada no paradigma PON. Este foi proposto como um novo paradigma de desenvolvimento de software, que apresenta algumas vantagens quando comparado aos paradigmas tradicionais (mais especificamente, o PI – Paradigma Imperativo - e o PD – Paradigma Declarativo) no que diz respeito ao seu modelo

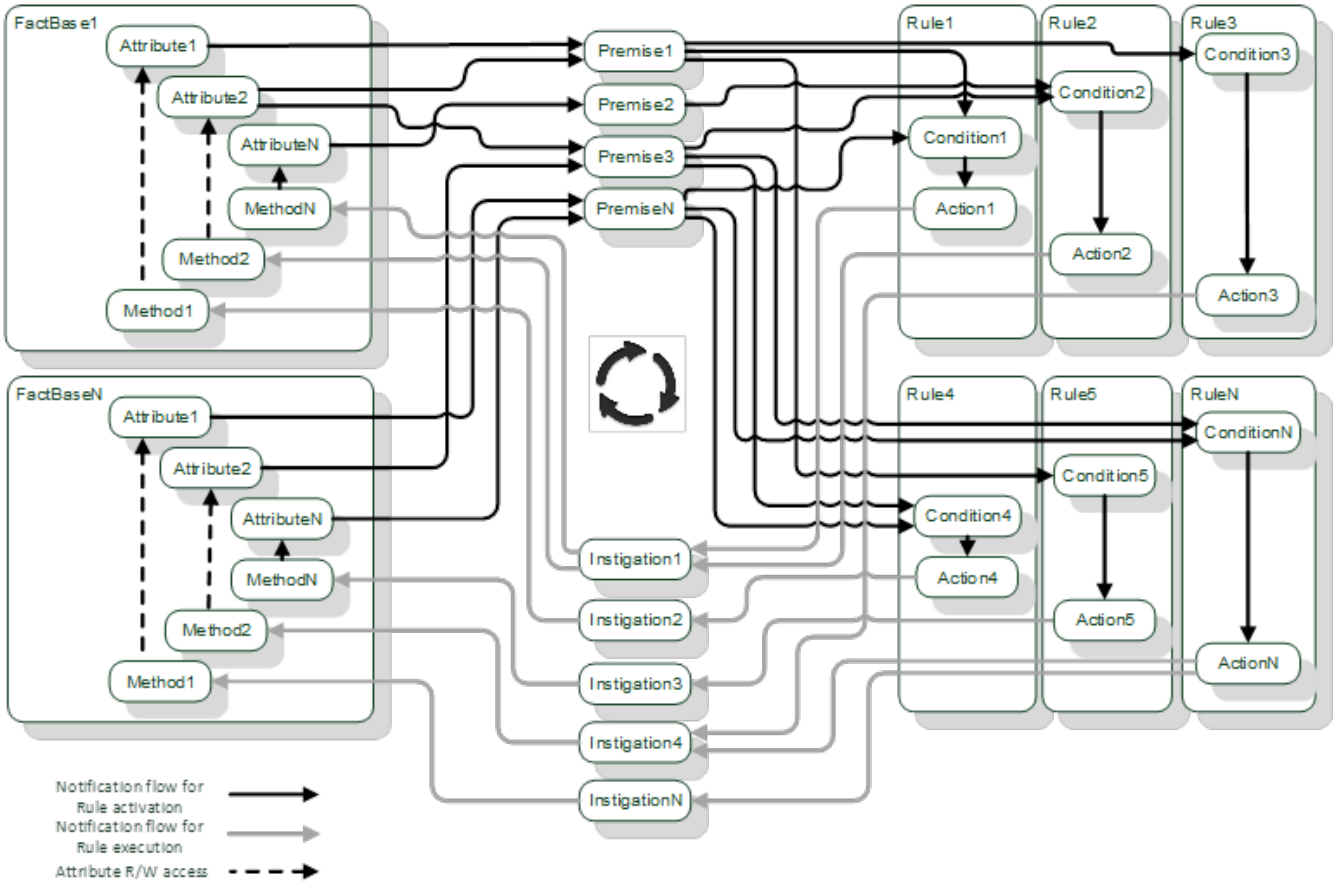

Figura 1. Colaboração por notificações das entidades do PON [10]

de execução, expressão lógico-causal e facto-execucional. Tais vantagens são constituídas por uma maior facilidade na concepção de sistemas que apresentem paralelismo ou distribuição, além da redução ou eliminação de alguns dos problemas clássicos de software PI e PD, como redundâncias de execução e acoplamento excessivo entre entidades computacionais, sem perder as vantagens da programação lógica, como clareza de codificação e coesão [11].

Em termos de expressão o PON pode ser utilizado na forma de um sistema orientado a 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 utilizados para representar objetos do mundo (real ou abstrato) em um sistema computacional, por meio de estados (atributos) e serviços (métodos). Os elementos *Rules*, por sua vez, definem o cálculo lógico-causal a ser efetuado sobre os estados dos *FBEs*, controlando a execução dos seus serviços. A colaboração entre estes elementos ocorre por meio de notificações diretas, que é um processo de inferência essencialmente distinto dos processos utilizados em software PI e em Sistemas Baseados em Regras (SBR) do PD [12]. O diagrama apresentado na Figura 1 demonstra graficamente os relacionamentos entre as entidades aqui citadas.

Nos *FBEs* os estados (atributos) são representados por meio de instância do tipo *Attribute*, enquanto os serviços dessas entidades (métodos) são representados por meio de instâncias do tipo *Method*. O elemento que representa uma *Rule* é composto por uma condição, representada pela instância do tipo *Condition* e uma Ação, pela instância do tipo *Action*. A instância do tipo *Condition* efetua um cálculo relacional ou causal sobre o valor de uma ou duas premissas, representadas por instâncias do tipo *Premise*. Estas são responsáveis por efetuar cálculo lógico-relacional sobre os valores dos atributos de um *FBE*, encapsulados em instâncias do tipo *Attribute*. A instância do tipo *Action*, por sua vez, referencia um ou mais instâncias do tipo *Instigation*, por meio das quais é capaz de disparar métodos dos *FBEs*, encapsulados em instâncias do tipo *Method* [12].

A partir do paradigma PON, esforços no sentido de materializar soluções foram apresentadas tanto para hardware [13] como para software. Nesta última é possível citar como principais contribuições: *framework* C++ [12] [9], ferramenta *wizard* [14] e a primeira versão de um compilador [7]. Atualmente, existe a segunda versão do compilador [15] o

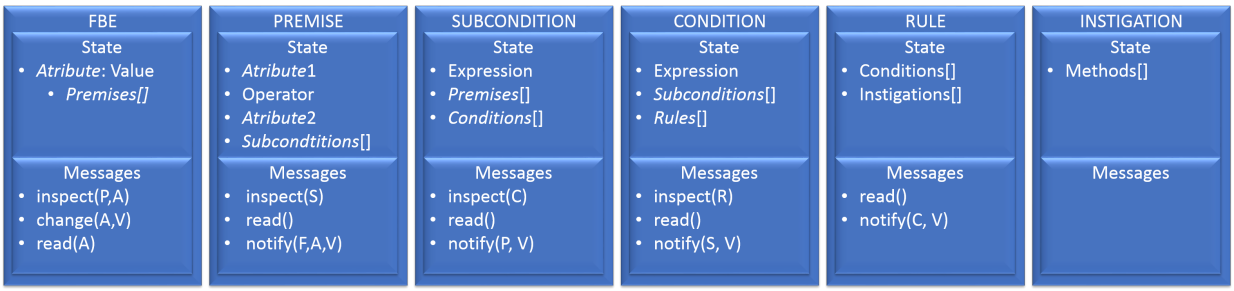

Figura 2. Modelagem alto nível dos elementos PON em atores

qual gera um grafo como uma fase intermediária de compilação. Desta forma, novas materializações são facilitadas, pois toda a etapa de validação da linguagem, estruturação e ligação de objetos é resolvida nesta etapa.

B. Erlang

O Erlang surgiu em meados da década de 1980 nos laboratórios de Ciência da Computação da Ericson com o intuito de resolver problemas *time-to-market* por Joe Armstrong, Robert Virding e Mike Williams. Eles construíram o Erlang sob a supervisão de Bjarne Däcker. Contudo somente em 1995 a linguagem Erlang considerada madura para ser usada em grandes projetos e em dezembro de 1998 ela foi liberada como código aberto sob licença EPL [8].

Concorrência em Erlang é fundamental para o seu sucesso. Em vez de fornecer encadeamentos que compartilham memória, cada processo Erlang é executado em seu próprio espaço de memória e possui seu próprio heap e pilha. Processos não podem interferir uns com os outros inadvertidamente, como é muito fácil em modelos de threading, levando a deadlocks e outros problemas comuns à programação concorrente. Processos se comunicam entre si via passagem de mensagens, onde a mensagem pode ser qualquer valor de dados Erlang. A passagem de mensagens é assíncrona, portanto, uma vez que uma mensagem é enviada, o processo pode continuar o processamento. As mensagens são recuperadas da caixa de correio (fila de mensagens recebidas pelo processo) do processo seletivamente, portanto, não é necessário processar mensagens na ordem em que são recebidas. Isso torna a simultaneidade mais robusta, principalmente quando os processos são distribuídos entre diferentes computadores e a ordem em que as mensagens são recebidas dependem das condições da rede ambiente [8].

O suporte a multiprocessamento simétrico (SMP) em Erlang foi primeiramente desenvolvido experimentalmente no final dos anos 90, e agora é parte integrante da versão padrão. O etos da equipe de desenvolvimento da Erlang / OTP na Ericsson é fazer o SMP funcionar, medir seu desempenho, encontrar os gargalos e otimizar. Desde o lançamento da primeira versão habilitada para SMP de Erlang, esta tem sido a sua abordagem. Em versões recentes, o modelo de máquina virtual evoluiu de uma única fila de

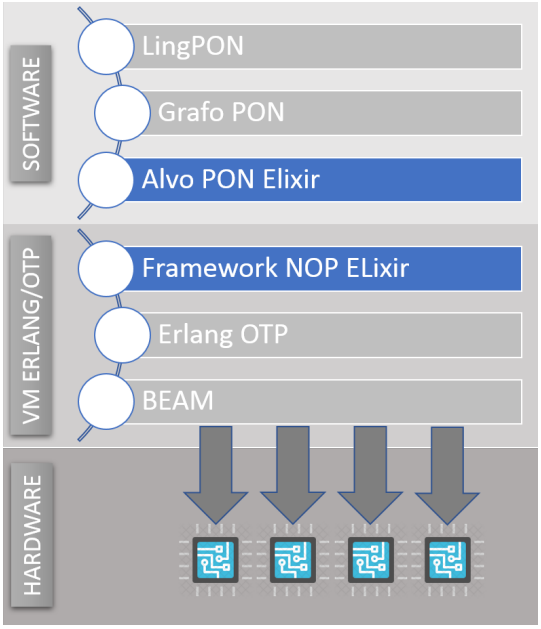

Figura 3. Arquitetura alto nível do LingPON multicore

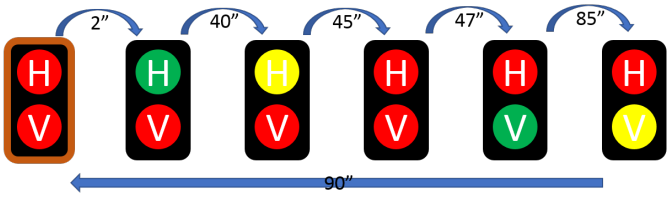

Figura 4. Diagrama de estados do CTA

execução monolítica - possivelmente com processos em execução em diferentes processadores - para uma fila de execução para cada processador, garantindo que a fila de execução não seja mais um gargalo para o sistema. À medida que surgem processadores mais complexos, o sistema de tempo de execução pode evoluir com eles [8].

III. Materiais e Métodos

A. PON e o modelo de atores

Erlang é conhecido como uma modelagem sob o paradigma de atores [8] conforme definido por Hewitt [16]

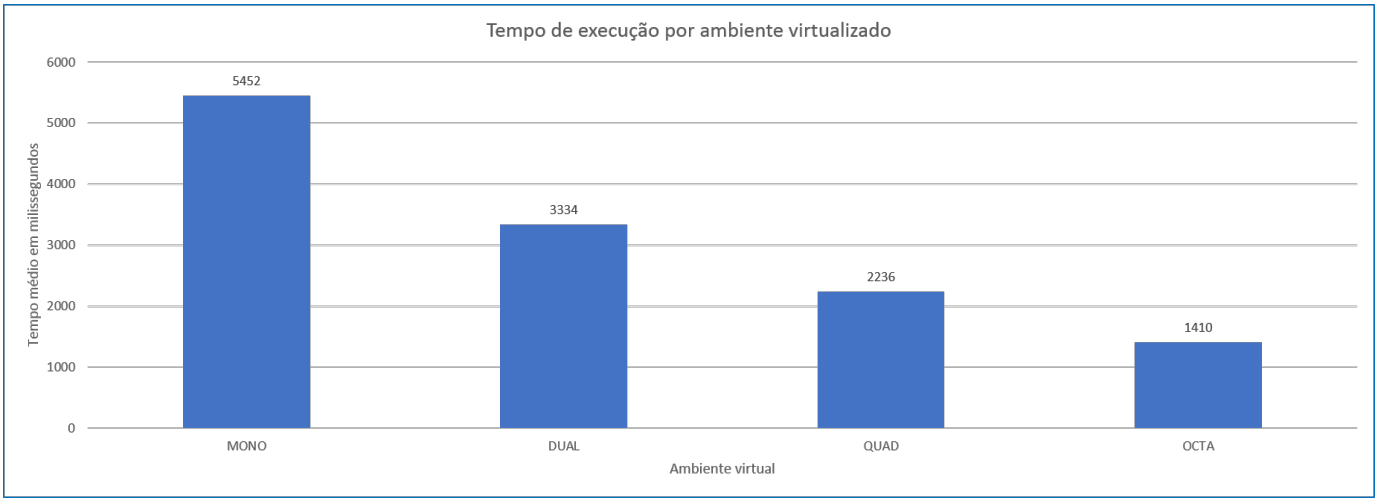

Figura 5. Resultados das simulações em ambientes EC2 tipo T2 Amazon AWS

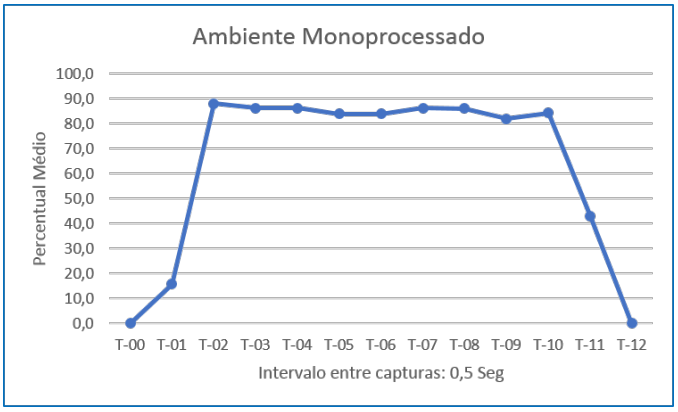

Figura 6. Taxa de ocupação do processador em ambiente monoprocessado

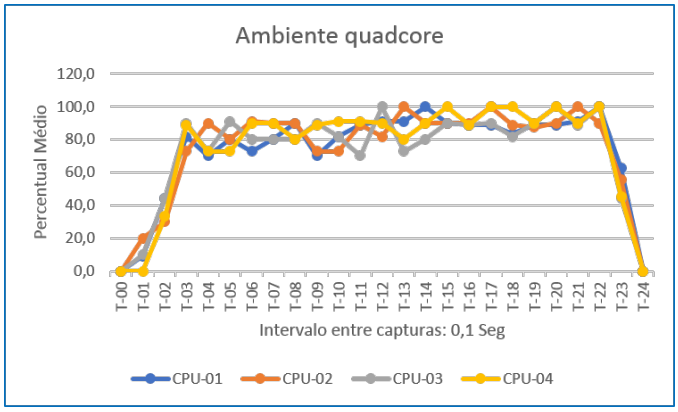

Figura 8. Taxa de ocupação dos processadores em ambiente *quad core*

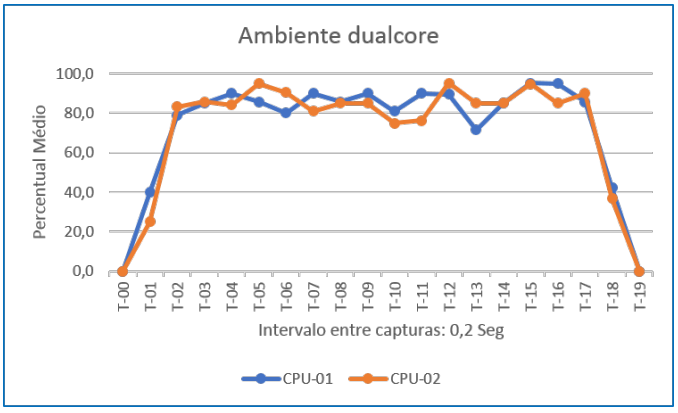

Figura 7. Taxa de ocupação dos processadores em ambiente *dual core*

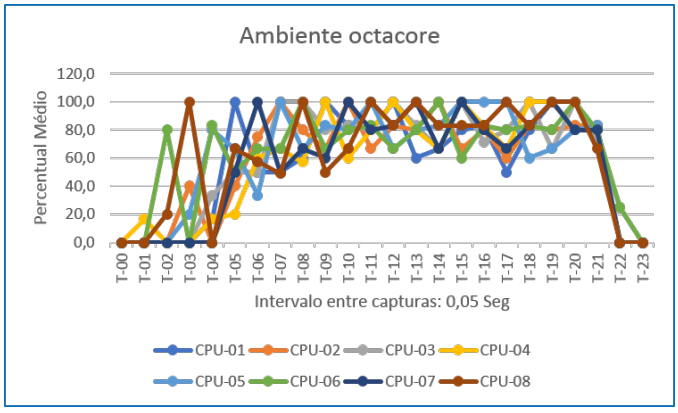

Figura 9. Taxa de ocupação dos processadores em ambiente *octa core*

[17] [18]. Portanto, o caminho para o aproveitamento da arquitetura concorrente Erlang passa pela aderência dos elementos do paradigma PON em processos com estados que se comunicam através de mensagens. Após uma análise minuciosa de cada elemento PON, foi possível chegar a uma modelagem alto nível de cada um de seus elementos em forma de micro ator. A Figura 2 apresenta um modelo alto nível da tradução dos elementos PON em atores de forma que possam ser então codificados e transformados em módulos Erlang.

B. Pacote nop_elixir

Uma vez passada a etapa de análise e aderência dos elementos do PON para elementos com comportamentos de atores, seguiu-se a etapa de codificação dos mesmos. Nesta etapa optou-se pela utilização da linguagem Elixir. Esta foi escolhida por estar totalmente imersa na plataforma Erlang e ainda disponibilizar mecanismos essenciais para a programação estruturada [19]. O pacote pode ser utilizado de maneira independente e está disponível para a comunidade Elixir para testes e avaliações. O resultado desta etapa é apresentado em forma de pacote HEX [20] para facilitar sua distribuição e utilização.

C. Compilador LingPON

Por fim, mas não menos importante, seguiu-se a etapa de fusão do compilador LingPON com o *framework* nop_elixir através de uma materialização do grafo PON [15]. O desenvolvimento desta integração entre grafo PON que levou cerca de dez horas de trabalho no total. Esta atividade compreendeu na identificação dos elementos no grafo para sua conversão elementos do *framework* e suas ligações devidamente alinhadas. O resultado final é uma arquitetura PON traduzida para atores sob a plataforma Erlang permitindo assim o total aproveitamento da capacidade de concorrência da máquina virtual BEAM. A figura 3 apresenta uma arquitetura em alto nível do resultado do trabalho de ligação do compilador LingPON com o hardware e seus núcleos através da arquitetrua Erlang e sua máquina virtual BEAM. As contribuições apresentadas neste artigo são apresentadas em destaque.

IV. Resultados e discussão

A. Experimento CTA

Para avaliar a capacidade de processamento concorrente multicore propôs-se como experimento um programa em LingPON para controle de trânsito automatizado (CTA). Seu objetivo é alternar os estados de um semáforo durante um ciclo de noventa segundos conforme descrito na Figura 4 - Os sinais com a letra *H* representam os estados dos semáforos para as ruas *horizontais* e os sinais com a letra *V* representam os estados dos semáforos para as ruas *verticais*. 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.

Uma vez finalizado o desenvolvimento, o próximo passo é o de testes de campo. Para isto foi utilizado um conjunto de quatro ambientes virtualizados do tipo T2 Ubuntu Server 14.04 LTS 64 bits e armazenamento SSD disponibilizados pela Amazon [21]. Estes foram escolhidos pela facilidade de acesso e disponibilização, bem como a possibilidade de se construir ambientes virtualizados com a mesma configuração de hardware variando-se apenas o número de núcleos, estes com mesma capacidade e velocidade de processamento [22].

Os resultados podem ser vistos na Figura 5. O primeiro gráfico à esquerda representa o tempo médio de execução em cada ambiente virtualizado. Como é possível perceber no cenário apresentado, há uma substancial redução de tempo de execução do programa de simulação conforme aumenta o número de núcleos disponíveis. Ao passo que os gráficos a direita demonstram uma distribuição balanceada entre os núcleos 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.

V. Conclusão e trabalhos futuros

Como é possível verificar, conforme o mesmo programa é executado em ambiente com múltiplos núcleos (*multicore*) o seu tempo de execução reduz ao passo que os núcleos permanecem todos com balanceamento de utilização durante todo o processamento, sugerindo que a execução está conseguindo se ocupar de todos os núcleos. Todo este aproveitamento alcançado com programação alto nível e estruturada, sem interferência ou conhecimento do programador em ambientes *multicore* graças ao desacoplamento da linguagem LingPON aliado à arquitetura concorrende Erlang. Somente com Erlang, tal concorrência somente seria possível com o desenvolvimento de alguma técnica de programção paralela, algo que ficou totalmente transparente graças à camada LingPON.

Como trabalhos futuros sugere-se avaliações mais profundas em relação à utilização em estresse dos núcleos, pois como visto no artigo, os núcleos tiveram sempre com balanceamento de atividades bem equlibrados porém, conforme o número de núcleos aumenta, os núcleos ficam menos ocupados sugerindo que ainda é possível ampliar a utilização destes processamentos. Outro trabalho futuro sugerido é o de aprimorar a linguagem LingPON para garantir determinismo não explorado nestes experimentos [23].

Referências

[1] G. Moore, "Cramming more components onto integrated circuits," *Eletronics Magazine*, 1965.

[2] S. Borkar and A. A. Chien, "The future of microprocessors," pp. 67–77, 2011.

[3] F. Eschmann, R. M. B. Klauer, and K. Waldschmidt, "Sdaarc : An extended cache-only memory architecture," pp. 62–70, 2002.

[4] K. Asanovic, R. Bodik, J. J. G. B. C. Catanzaro, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," 2006.

[5] S. H. Kaisler, *Software Paradigms*.

[6] B. Stroustrup, *The C++ Programming Language. 4th Edition*.

[7] C. A. Ferreira, "Linguagem e compilador para o paradigma orientado a notificações (PON): avanços e comparações," Dissertação. Mestrado em Computação Aplicada - PPGCA. Universidade Tecnológica Federal do Paraná (UTFPR) , Curitiba, p. 227, Agosto 2015.

[8] F. Cesarini and S. K. Thomson, *Erlang Programing*.

[9] A. F. Ronszcka, "Contribuição para a concepção de aplicações no paradigma orientado a notificações (PON) sob o viés de padrões," Dissertação. Mestrado em Engenharia Elétrica e Informática Industrial - CPGEI. Universidade Tecnológica Federal do Paraná (UTFPR) , Curitiba, p. 236, Agosto 2012.

[10] R. D. Xavier, "Paradigmas de desenvolvimento de software: comparação entre abordagens orientada a eventos e orientada a notificações," Programa de Pós-Graduação em Computação Aplicada (PPGCA), 2014.

[11] J. G. Brookshear, *Computer Science: An Overview*.

[12] R. F. Banaszewski, “Paradigma Orientado a Notificações: Avanços e Comparações,” Dissertação. Mestrado em Engenharia Elétrica e Informática Industrial - CPGEI. Universidade Tecnológica Federal do Paraná (UTFPR) , Curitiba, p. 261, Março 2009.
[13] R. Kerschbaumer, J. M. Simão, R. R. Linhares, P. C. Stadzisz, and C. R. E. Lima, “Paradigma Orientado a Notificações para a Síntese de Lógica Reconfigurável,” *Proceedings of III Brazilian Congress on Fuzzy Systems (III CBSF)*, Outubro 2015.
[14] G. Z. Valença, “Contribuição para materialização do paradigma orientado a notificações (PON) via framework e wizard,” Dissertação. Mestrado em Computação Aplicada - PPGCA. Universidade Tecnológica Federal do Paraná (UTFPR) , Curitiba, p. 205, Agosto 2012.
[15] A. F.Ronszcka, “Método para a criação de linguagens e compiladores voltados a plataformas distintas baseado no PON,” Dissertação. Mestrado em Computação Aplicada - CPGEI. Universidade Tecnológica Federal do Paraná (UTFPR) , Curitiba, p. 227, Maio 2019.
[16] C. Hewitt, P. Bishop, and R. Steiger, “A universal modular actor formalism for artificial intelligence,” 1973.
[17] C. Hewitt, “Viewing control structures as patterns of passing messages,” 1977.
[18] C. Hewitt and H. Baker, “Laws for communicating parallel processes,” 1977.
[19] D. Thomas, *Programing Elixir*.
[20] F. Negrini. Nop (notification oriented programing) simulator for elixir. [Online]. Available: https://hex.pm/packages/nop_elixir
[21] Instâncias t2 no amazon ec2. [Online]. Available: https://aws.amazon.com/pt/ec2/instance-types/t2/
[22] Amazon ec2 - tipos de instâncias. [Online]. Available: https://aws.amazon.com/pt/ec2/instance-types/#burst
[23] J. M. Simão and P. C. Stadzisz, “Mecanismo de resolução de conflito e garantia de determinismo para o paradigma orientado a notificações (pon),” 2010.