Applying Parallelization Strategies for Inference Mechanisms Performance Improvement

D. Araújo, A. Hentges, S. Rigo, and R. Righi

***Abstract*—The use of semantics technologies for system development is increasing nowadays. The knowledge representation in ontologies can be used in a lot of applications, ranging from knowledge-based recommendation systems until the Semantic Web applications. The core component of the semantic applications is the logical inference engine, which process and generates new facts into the knowledge base from production rules. The inference engine´s performance is directly related to the length of the knowledge base and the necessities of the nowadays knowledge bases are becoming a challenge. This paper presents a knowledge base´s search algorithm for the RETE Algorithm which uses the intrinsic parallel structures from the modern computers, augmenting the performance of the inference engine. We implemented a Threads based and a GPU based search engine and compared their performance. We point as main contributions of this paper the parallel system that implements the search engine and the algorithm for vectorization of the knowledge base.**

***Index Terms*—RETE Algorithm, Inference, GPU, Parallel Computing, Threads.**

I. Introdução

ATUALMENTE existe uma grande difusão no uso de aplicações baseadas em tecnologias semânticas. O desempenho no acesso e manipulação dos dados de grandes bases de conhecimentos é um dos principais fatores para o funcionamento adequado da Web Semântica e de sistemas de recomendação, por exemplo. A possibilidade de criar novos conhecimentos a partir da inferência sobre os dados é uma das principais características dos sistemas semânticos.

No entanto, o processo de inferência geralmente é altamente dispendioso em termos de recursos computacionais. Sistemas baseados na inferência lógica sobre ontologias para processamento de linguagem natural [1]–[3] ou para identificação de significado [4]–[6] tem grande dependência da velocidade de processamento e necessitam de rapidez em seu processamento.

Existem hoje tecnologias disponíveis para apoiar a melhoria de desempenho do processo de inferência dos sistemas semânticos, as quais podem ser utilizadas diretamente em conjunto com a arquitetura do motor de inferência lógico. Uma possibilidade seria, por exemplo, a distribuição do processo de raciocínio em um cluster computacional [7]–[9].

Araujo, D. A, UNISINOS University, Brazil (denis.andrei.araujo@gmail.com).
Hentges, A. R. UNISINOS University, Brazil (alencarhentges@gmail.com).
Rigo, S. J. UNISINOS University, Brazil (rigo@unisinos.br).
Righi, R. R. UNISINOS University, Brazil (rrrighi@unisinos.br).

Entretanto, o uso dos recursos intrínsecos dos computadores atuais (tais como múltiplos núcleos de processamento e dispositivos internos de paralelização massiva) para apoiar melhorias de performance em sistemas semânticos é um tema que demanda pesquisa e aprofundamento, para identificar o aproveitamento mais efetivo destas possibilidades, devido às peculiaridades do funcionamento dos sistemas de inferência.

Neste artigo são apresentadas duas abordagens para otimização do desempenho no tempo de busca de padrões em bases de conhecimento, um dos processos essenciais na operação dos mecanismos de inferência usados em sistemas semânticos. As abordagens apresentadas neste artigo fazem uso de duas estratégias diferentes de paralelização, sendo a primeira com base na exploração da paralelização de *threads* [35] nas *Central Process Units* (CPU) atuais e a segunda baseada em dispositivos de processamento paralelo massivo denominados de *Graphics Processing Unit* (GPU) [10].

Serão analisados os resultados obtidos na comparação do desempenho em relação ao sistema de busca do motor de inferência do *framework* Jena [18], que é largamente utilizado para a implementação de sistemas semânticos. A pesquisa aqui relatada faz parte de um projeto mais amplo, o qual tem como objetivo o desenvolvimento e disponibilização à comunidade acadêmica de um mecanismo de inferência de alto desempenho baseado em regras genéricas para processamento de bases de conhecimentos extensas.

Considera-se como principais contribuições deste artigo a apresentação de uma nova metodologia para o desenvolvimento de um motor de consulta que possibilita a otimização do tempo de busca de dados para sistemas baseados em tecnologia semântica pela exploração de recursos intrínsecos de paralelização dos computadores modernos, bem como um algoritmo de vetorização da base de conhecimento, ambos visando o incremento de desempenho de motores de inferência para processamento de grandes bases de dados.

Este artigo está assim organizado: primeiramente apresentam-se os trabalhos correlatos e em seguida é apresentada a fundamentação teórica. A seguir, é descrita a abordagem proposta para uso do paralelismo no motor de busca do mecanismo de inferência, sendo então descrita a metodologia de avaliação e analisados os resultados obtidos em experimentações. Por fim, são apresentadas as considerações gerais e projeções de trabalhos futuros.

II. Trabalhos correlatos

O uso de programação paralela para a aplicação de regras de produção, para a implementação da semântica *Resource Description Framework* (RDF) ou *Web Ontology Language*

(OWL), por exemplo, para inferência sobre extensas ontologias de centenas de milhões de triplas, é foco de interesse atual de pesquisadores da área de programação semântica, particularmente pelo uso de processamento baseado em cluster computacional [11].

Um motor de inferência baseado no Hadoop [12] chamado WebPie [8, 13] codifica as regras necessárias como um conjunto de operações *Map and Reduce* e as executa com a distribuição de carga gerenciada pelo sistema Hadoop. Há outros trabalhos que propõem soluções baseados no modelo de programação MapReduce [14] para a paralelização da busca e execução das regras de produção com base em cluster computacional [15]–[17]. Os dados são distribuídos em lotes menores entre processos que rodam em uma rede de nós computacionais. Conforme já comentado, este modelo de computação paralela baseada em cluster apresenta como desvantagem a complexidade para implementação para o tratamento dos novos dados produzidos e um custo maior de implementação.

Há também as abordagens que visam tirar proveito de estruturas de paralelização em computadores individuais, explorando suas estruturas de paralelização de CPU com vários núcleos e GPUs massivamente paralelas. Abordagens tais como a empregada no mecanismo de inferência ELK [18] são um exemplo de sistemas que buscam melhoria de desempenho explorando os recursos intrínsecos de paralelização do processamento.

Este sistema apresenta um motor de inferência que implementa uma sintaxe mais concisa da Lógica de Descrição (DL) para representar os axiomas OWL, a qual suporta axiomas de inclusão de conceitos (TBox) mas não asserções (ABox). Esta característica restringe o uso do ELK a um grupo de ontologias de expressividade mais restrita, além de implementar uma semântica específica.

O uso de processamento massivamente paralelo das GPUs não é indicado para todo e qualquer problema, impondo algumas restrições no tratamento dos dados [19]. Embora seja uma abordagem de alto desempenho para cálculos repetitivos sobre grandes conjuntos de dados numéricos, é bastante restrita a sua utilização no processamento simbólico dos dados [20], o que é uma necessidade fundamental das aplicações semânticas, pois estas lidam fundamentalmente com necessidades de massivo processamento simbólico de dados [21].

No trabalho apresentado inicialmente em [22], e mais recentemente em [23], vê-se uma proposta de pré-processamento dos dados para facilitar o uso dos recursos de paralelismo. Neste pré-processamento tem-se em vista primeiramente a conversão de dados simbólicos em dados numéricos e posterior vetorização destes dados, a fim de favorecer o seu processamento paralelo nas GPUs. Estes trabalhos influenciaram no desenvolvimento do motor de busca apresentado neste artigo.

Os trabalhos estudados, com foco na exploração do paralelismo massivo para processamento de grandes bases de conhecimento, apresentam ainda aspectos incipientes quanto à otimização de aspectos de busca de informações. Como este é um aspecto crucial para a obtenção de uma alta performance dos sistemas de processamento semântico, trata-se de uma lacuna de pesquisa atualmente. A abordagem proposta neste artigo apresenta contribuições no algoritmo de busca visando otimização do desempenho, que permitem ampliar os aspectos de performance gerais.

III. Fundamentação Teórica

O algoritmo RETE [24] tem papel fundamental no desempenho do sistema de inferência proposto neste trabalho, sendo por isso apresentado na próxima seção. Em seguida, são tecidos comentários sobre recursos de programação paralela utilizados nos testes para avaliação do modelo proposto.

A. O Algoritmo RETE

O tempo de execução de um mecanismo de inferência baseado em regras depende principalmente da velocidade de identificação dos dados que disparam as regras de produção. Um algoritmo amplamente utilizado para a implementação de motores de inferência baseado em regras é o algoritmo RETE [24]. No cerne deste algoritmo encontra-se um elemento de pré-processamento dos dados cujo principal objetivo é a rápida localização prévia de todos os dados que atendem os padrões estipulados nas regras de produção.

O Algoritmo RETE consiste na construção de uma rede de nós que representam os padrões de busca das regras. Genericamente, dá-se o nome de *Left Hand Side* (LHS) a estes padrões de busca das regras. No caso dos sistemas baseados em tecnologia semântica, estes consistem de padrões simples ou compostos de uma ou mais triplas de dados RDF [25].

Os padrões simples são compostos de uma única tripla e são representados na rede do Algoritmo RETE por estruturas de dados denominadas de nós alfa. Já os padrões compostos de duas ou mais triplas são representados pela combinação de dois ou mais nós alfa em nós denominados beta.

Os padrões de busca simples são representados sempre por três elementos entre parênteses, podendo conter variáveis, as quais são representadas por um nome prefixado pelo sinal de interrogação.

O principal diferencial do Algoritmo RETE é a rapidez de processamento e disparo das regras, baseado na análise prévia dos dados. Para alcançar este desempenho, uma lista com todas as triplas de dados que podem ser associadas com a regra é armazenada em cada nó (alfa ou beta). Esta lista é criada pela propagação dos dados de entrada na rede de nós do Algoritmo RETE.

Visando apresentar um maior detalhamento do algoritmo, será demonstrado um exemplo com a rede de nós que representa as regras apresentadas na Tabela I.

Tabela I
Exemplo De Regras De Produção

(?s ?p ?o) → (?p rdf:type rdf:Property)	(R1)
(?s ?p ?o) (?p rdfs:domain ?c) → (?s rdf:type ?c)	(R2)

Como pode ser visto na Figura 1, a rede resultante das regras apresentadas na Tabela I tem um total de três nós: dois nós alfa

e um beta. O nó α1 representa o padrão de busca (?s ?p ?o) e o nó α2 indica o padrão de busca (?p rdfs:domain ?c). O nó β1 representa a união dos nós α1 e α2. Uma regra completa é representada por um único nó final. Na rede representada na Figura 1, a regra R1 tem α1 como seu nó final, enquanto R2 tem β1 como nó final.

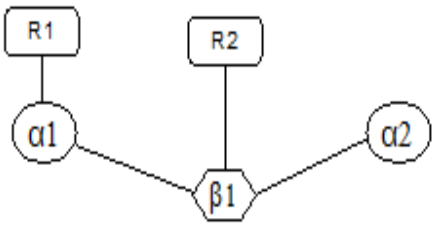

Fig. 1. Rede de nós resultante das regras R1 e R2.

Prosseguindo com o exemplo, considere-se as triplas apresentadas na Tabela II como dados de entrada.

A propagação dos dados de entrada da Tabela II resultaria na rede apresentada na Figura 2. As triplas WM1, WM2 e WM3 atendem ao padrão estabelecido no nó α1, pois este é composto por três variáveis. Já para o nó α2, somente a tripla WM3 atende ao seu padrão. No caso do nó β1, temos os pares {WM1, WM3} e {WM2, WM3} como dados que atendem aos padrões da LHS da regra R2.

TABELA II
DADOS DE ENTRADA PARA A REDE

Maria parentOf José	(WM1)
Pedro parentOf João	(WM2)
parentOf rdfs:domain Parent	(WM3)

O nó α1 é nó final da regra R1 e o nó β1 é nó final da regra R2. Desta forma, verifica-se que os dados de entrada farão com que a regra R1 dispare três vezes, produzindo três novas triplas, uma delas duplicada e por isto considerada somente uma vez.

A regra R2 é disparada duas vezes e produz duas novas triplas. Ao final deste primeiro ciclo de execução, a memória de trabalho terá quatro novas triplas, as quais são apresentadas na Tabela III.

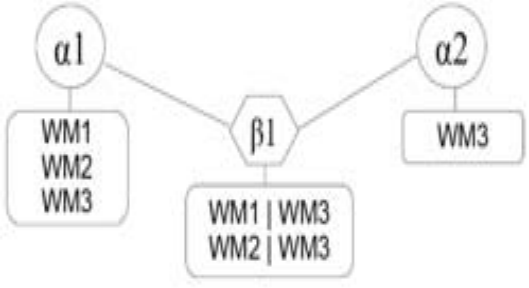

Fig. 2. Rede com dados propagados.

O mesmo processo repete-se novamente para os novos dados produzidos, até que não ocorra mais o disparo de nenhuma regra, finalizando assim a execução das mesmas.

TABELA III
TRIPLAS GERADAS PELO ALGORITMO RETE

parentOf rdf:type rdf:Property	(WM4)
rdfs:domain rdf:type rdf:Property	(WM5)
Maria rdfs:type Parent	(WM6)
Pedro rdfs:type Parent	(WM7)

Observa-se que o processo de execução das regras depende da busca de todos os dados que atendem aos padrões LHS dos nós alfa e posteriormente dos nós beta. Em um primeiro momento, está busca por triplas que atendam os padrões ocorre em toda a base de conhecimentos, testando cada uma das triplas armazenadas para bases de conhecimento extensas, com milhões ou até mesmo bilhões de triplas, o processo de busca torna-se, portanto, uma etapa crítica para o bom desempenho do algoritmo como um todo.

A abordagem proposta neste artigo busca atender exatamente esta fase crítica da inferência dos sistemas baseados em tecnologia semântica, com a otimização do tempo de localização dos padrões de busca das LHS na base de conhecimento através da paralelização massiva deste processo.

B. Recursos de Programação Paralela

Os computadores atuais contam com estruturas internas para otimização do tempo de execução através da paralelização do processamento. Dentre as opções atualmente disponíveis para processamento paralelo, destaca-se a possibilidade do uso dos recursos intrínsecos das unidades de processamento dos computadores atuais (as CPUs) e de dispositivos que contém uma alta concentração de unidades paralelas de processamento (as denominadas GPUs). A exploração da capacidade de processamento paralelo de computador individualmente é um foco de estudo que vem recebendo a atenção da comunidade científica em variadas áreas de pesquisa [26]–[32]. Nas pesquisas que buscam a otimização do desempenho dos sistemas baseados em tecnologia semântica também é visível o interesse em explorar-se a capacidade de processamento paralelo [22], [23], [33], [34].

Conforme citado anteriormente, os experimentos relatados neste artigo visam a verificação do desempenho de dois modelos de paralelização de processos para otimização do tempo de busca em bases de conhecimentos: a programação baseada em *threads* [34] e o paralelismo massivo proporcionado pelas unidades gráficas de processamentos denominadas de GPU.

Tradicionalmente, os *threads* são definidos como unidades de execução dentro de um processo. Quando múltiplos programas executam concorrentemente em um único espaço de endereços, diz-se que este é um sistema *multi-thread* [35]. A programação *multi-thread* é um paradigma de programação reconhecidamente importante, principalmente devido a necessidade crescente de arquiteturas de sistemas multiprocessados.

Mais recentemente, com a maior disponibilidade de dispositivos projetados especificamente para processamento paralelo e a custos razoáveis, observou-se o início de uma

grande quantidade de estudos sobre a aplicação destes dispositivos para a resolução de uma ampla gama de aplicações, incentivando assim o investimento por parte dos fabricantes de tais dispositivos a disponibilizarem frameworks de desenvolvimento de aplicações que explorem ao máximo as capacidades de processamento paralelo [19].

Dentre os frameworks atualmente disponíveis para implementação de sistemas baseados em GPU, destaca-se o OpenCL, um framework de programação heterogêneo que possibilita o desenvolvimento de aplicações que podem ser executadas em uma grande variedade de tipos de dispositivos e suporta uma alta gama de tipos de paralelismo.

O framework OpenCL possuí diferentes implementações, as quais geram quantidades diferentes de *threads* internamente para suportar diferentes recursos. Além disso, o framework permite que se explore tanto as estruturas internas das CPUs para a execução de processos em paralelo, quanto os núcleos de processamento paralelo das atuais GPUs, configurando assim uma grande flexibilidade ao sistema, que pode então rodar processos paralelos tanto em computadores com CPUs de múltiplos núcleos quanto naqueles equipados com GPUs.

IV. Motor De Busca Baseado Em Processamento Paralelo

Os desenvolvimentos de sistemas que se baseiam em tecnologias semânticas são normalmente implementados sobre um framework que simplificam o acesso e processamento das bases de conhecimento. Dentre os frameworks semânticos mais utilizados atualmente, encontra-se o Jena [18], uma biblioteca de classes desenvolvida em Java e amplamente utilizada para o desenvolvimento de sistemas semânticos.

O projeto de pesquisa no qual este trabalho se encontra tem como objetivo o desenvolvimento e disponibilização à comunidade de um mecanismo de inferência de alto desempenho para o framework Jena.

Por conta deste objetivo, buscou-se bibliotecas baseadas em Java para a implementação do motor de busca que vai realizar a carga inicial dos dados na rede do RETE. Esta carga inicial dos dados na rede do Algoritmo RETE é o processo mais crítico em relação ao tempo de execução dos sistemas semânticos baseados no Jena.

Para o desenvolvimento do protótipo do motor de busca baseado em *threads*, foi utilizada a implementação da biblioteca padrão do Java chamada *Threads* [36], uma classe nativa do Java que implementa o paradigma de *threads* de programação na linguagem Java. Para o motor de busca baseado em GPU, utilizou-se o JavaCL o qual é uma API Java para chamada de serviços fornecidos pelo driver do OpenCL.

Levando-se em consideração o objetivo de comparar o modelo de programação paralela baseado em *threads* com o modelo baseado em GPU, foi projetado um modelo de dados que permitisse realizar este comparativo da forma mais imparcial possível, sem favorecer um ou outro modelo de programação.

Tendo esta diretriz em vista e levando-se em consideração que o modelo de programação do OpenCL tem grandes restrições para o processamento de dados não numéricos, ficou definido que as triplas da base de conhecimento seriam representadas por um identificador numérico único e individual, possibilitando assim o seu armazenamento em vetores para processamento otimizado tanto no paradigma de programação das GPUs quanto no baseado em *threads*.

A este processo de indexação numérica da base de conhecimento deu-se o nome de vetorização das triplas. Para melhor entendimento deste processo de vetorização, apresenta-se na Tabela IV um exemplo simplificado de vetorização de triplas. Nele pode ser observado como o elemento "parentOf" é indexado com o número 2 e como este número é reutilizado nas três triplas, visto que o elemento se repete.

Tabela IV
Exemplo de Vetorização de Triplas

Tripla original	Tripla Vetorizada
Maria parentOf José	(1, 2, 3,
Pedro parentOf João	4, 2, 5,
parentOf rdfs:domain Parent	2, 6, 7)

A triplas originais são indexadas e inseridas em um único vetor de inteiros, cada uma representada por um número identificador unívoco. Para o exemplo apresentado acima, o vetor de triplas ficaria assim: (1, 2, 3, 4, 2, 5, 2, 6, 7).

Também é necessário realizar a vetorização dos padrões de busca das regras, as quais seguem um procedimento diferente. No caso da vetorização dos padrões de busca das regras, temos a possibilidade de haver elementos "livres", os quais são chamados de variáveis. No processo de vetorização do LHS das regras, as variáveis são substituídas pelo valor zero, como pode ser visto no exemplo da Tabela V.

Tabela V
Exemplo de Vetorização das Regras

Regra original	LHS Vetorizado
(?s ?p ?o) → (?p rdf:type rdf:Property)	(0 0 0)
(?s ?p ?o)	(0 0 0)
(?p rdfs:domain ?c) → (?s rdf:type ?c)	(0 6 0)

Após ter sido vetorizada toda a base de conhecimento e todos os padrões de busca, é possível realizar a busca paralela, tanto no paradigma baseado em *threads* quanto no baseado em GPU. Embora os paradigmas de programação sejam bastante diferentes entre si, o algoritmo de busca pode ser exatamente o mesmo, pois vai realizar uma busca sequencial de comparação de padrões com as triplas, seguindo o algoritmo apresentado na Figura 3.

```
for(i=0; i < TOTAL_TRIPLAS; i=i+1){
    CASOU[i] = (( S[i]-LHS[0] ) * LHS[0] )
             +(( P[i]-LHS[1] ) * LHS[1] )
             +(( O[i]-LHS[2] ) * LHS[2] );
}
```

Fig. 3. Algoritmo de busca para triplas vetorizadas.

Ao final da execução deste algoritmo simplificado descrito na Figura 3, o vetor CASOU, conterá zero se, e somente se, houve uma igualdade (um casamento) entre o padrão de busca e a i-ésima tripla da base de conhecimentos.

Observa-se que na abordagem proposta aqui, não há nenhuma dependência de dados para a busca de triplas que casem com o padrão do LHS. Deste modo este algoritmo de comparação pode ser aplicado de forma paralela sobre toda a base de conhecimento, uma vez que cada processo concorrente se ocupa de comparar o padrão de busca do LHS com uma tripla específica e diferente das demais existentes na base de conhecimento.

Esta abordagem dispensa o uso de sincronismo entre os processos paralelos, permitindo assim uma execução completamente paralelizada, tanto no paradigma de programação baseada em *threads*, quanto na baseada em processamento massivamente paralelo (GPU).

Então, para fins de comparação, implementou-se três motores de busca para preenchimento dos nós alfa e beta da rede RETE, utilizando-se três diferentes paradigmas de execução: sequencial, baseado em *Threads* e baseado em OpenCL. Na próxima seção são apresentados os desempenhos destas implementações.

V. Metodologia e Resultados

A metodologia escolhida para a verificação do desempenho dos algoritmos de busca foi realizada sobre dados artificiais gerados aleatoriamente utilizando semente 2354. A partir desta definição, desenvolveu-se um programa em Java cujo procedimento inicial consiste em gerar uma longa sequência de números aleatórios e armazená-los em um vetor. Esta sequência de números gerados tem como objetivo representar uma base de conhecimento já vetorizada.

O objetivo principal deste experimento é verificar o desempenho dos motores de busca implementados sob os paradigmas de programação sequencial e paralelo sobre bases de conhecimento muito extensas. Por isto foram geradas 3 bases de conhecimento vetorizadas contendo 30, 45 e 60 milhões de triplas. Uma vez obtida a base de conhecimento vetorizada (BCV), o sistema gera uma tripla de números, também escolhidos aleatoriamente, a qual será utilizada como o padrão de dados a serem localizados na BCV.

Uma questão importante a salientar é que na tripla de números do padrão de busca podemos ter ou não variáveis, ou seja, elementos cujo valor é indiferente para o motor de busca. Quando um padrão de busca contiver uma variável, isto será representado com o valor do elemento sendo igual a zero.

Por exemplo, um padrão de busca “90 22 0” implica que o motor de busca deve localizar todas as triplas da BCV nas quais o primeiro e o segundo elemento contém respectivamente os valores 90 e 22, não importando qual valor do terceiro elemento da tripla.

Os padrões de busca podem conter uma, duas, três ou até mesmo não ter de apresentar variável. Para cada uma destas possibilidades, foi executado um experimento em separado, a fim de verificar o desempenho do motor de busca em cada uma destas possíveis situações.

Então, para cada BCV gerada, sorteia-se um padrão de busca e realiza-se a sua localização, primeiramente sem variáveis, depois com uma, duas e finalmente com três variáveis. Cada busca é realizada vinte vezes seguidas e é registrado o seu tempo de execução médio nos paradigmas de programação sequencial e paralelo. Para o paradigma paralelo, implementou-se um motor de busca baseado em *Threads* e outro baseado em OpenCL.

De acordo com a metodologia descrita, foram executados experimentos com 30, 45 e 60 milhões de triplas, os quais foram executados em um computador com CPU Intel Core i7 3517U com 8 GB de memória RAM, contendo uma GPU NVIDIA GeForce GT 635M com 2GB de memória dedicada. Na primeira coluna das Tabelas VI até VIII apresenta-se o número de variáveis do padrão de busca, sendo o tempo de busca de cada paradigma de programação apresentado em milissegundos nas colunas subsequentes.

Tabela VI
Tempos de busca para 30 milhões de triplas

N° variáveis	Sequencial	*Thread*	OpenCL
0	1747	959	204
1	2770	1385	172
2	4018	2005	173
3	5800	2599	154

Tabela VII
Tempos de busca para 45 milhões de triplas

N° variáveis	Sequencial	*Thread*	OpenCL
0	2522	1522	254
1	5338	2124	235
2	7186	2984	247
3	8650	4006	248

Tabela VIII
Tempos de busca de 60 milhões de triplas

N° variáveis	Sequencial	*Thread*	OpenCL
0	4643	1961	312
1	5659	2735	313
2	8489	4306	348
3	11690	5542	451

Percebe-se claramente que, como previsto, os paradigmas de programação paralela efetivamente melhoram o desempenho do algoritmo de busca, proporcionando um ganho substancial da velocidade na busca.

Como já havia uma expectativa de melhor desempenho da programação paralela, planejou-se os experimentos com o objetivo principal de verificar quais dos paradigmas de programação paralela ofereceriam o melhor desempenho.

Se, por um lado constata-se que a paralelização baseada em *Threads* demonstrou ser em média aproximadamente 2.11 vezes mais rápida que a sequencial, por outro verifica-se que o

paradigma baseado em OpenCL obteve destaque apresentando em média um desempenho 22.12 vezes mais rápido que a abordagem sequencial.

Comparando-se o desempenho dos motores de busca que utilizam processamento paralelo, ou seja, dos paradigmas de programação baseadas em Thread e OpenCL, verifica-se que o OpenCL é em média 10.35 vezes mais rápido, o que demonstra sua superioridade em relação ao paradigma baseado em *Threads*.

Além disto, observa-se também que a exploração da capacidade de processamento massivamente paralelo das GPUs apresenta outras vantagens muito interessantes, as quais são comentadas nos próximos parágrafos.

Ao observar-se o desempenho em relação ao número de variáveis utilizadas nos padrões de busca, verifica-se que para os paradigmas sequencial e baseado em *Threads*, o desempenho da busca degrada conforme o maior número de variáveis.

No entanto, o motor de busca baseado em OpenCL tem comportamento inverso, apresentando desempenhos melhores em buscas com maior número de variáveis. Verifica-se nos gráficos apresentados na Figuras 4 que o OpenCL somente apresenta leve degradação de desempenho no experimento realizado com 60 milhões de triplas. Tal fato, levanta suposições quanto a problemas causados pela falta de memória do equipamento onde foram executados os experimentos, contudo não foi possível comprovar esta hipótese.

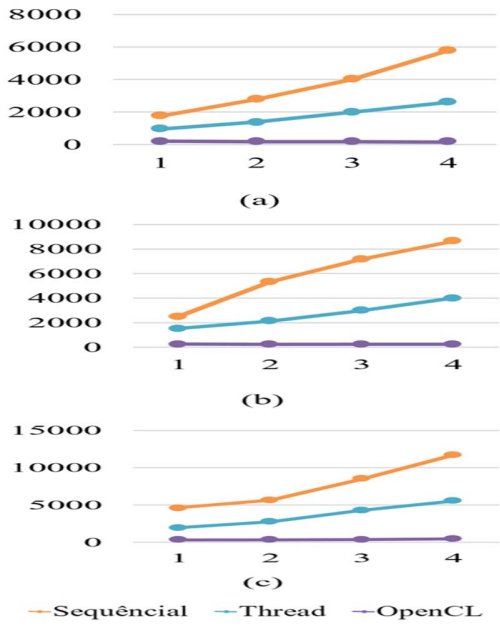

Fig. 4. Tempos de busca para (a) 30 milhões, (b) 45 milhões e (c) 60 milhões de triplas.

Em ambientes controlados, a ocorrência de consultas com conhecimento prévio das variáveis torna possível a simulação do acontecimento para averiguação da capacidade e tempo de resposta dos algoritmos. Entretanto, fora do ambiente controlado, a execução da procura tende a ocorrer com o desconhecimento de pelo menos uma das variáveis envolvidas no processo, já que uma das principais vantagens das regras de produção de conhecimento é sua aplicação genérica em diversos contextos. Sendo assim, na prática, o motor de busca baseado em OpenCL possuí um desempenho melhor frente aos demais algoritmos comparados.

Outra indicação de favorecimento do OpenCL em relação aos *Threads* diz respeito à aceleração da degradação de desempenho em relação ao tamanho da base de dados a processar. Verifica-se que o motor baseado em *Threads* fica em média 61.54% mais lento no decorrer das buscas realizadas nas três bases analisadas, enquanto que no OpenCL a degradação do desempenho foi de 59.76%.

Verifica-se que a degradação de desempenho das abordagens baseadas em Sequencial, *Threads*, OpenCL ficam muito próximas (61.76%, 61.54% e 59.46% respectivamente) levando em consideração todas as bases. Porém, quando observada apenas o repositório contendo 60 milhões de triplas, observa-se que a abordagem Sequencial obtém a menor degradação (22.26%) frente aos comparados *Threads* (26.87%) e OpenCL (30.9%)

Do ponto de vista dos objetivos do experimento e dos resultados aqui relatados, considera-se exitoso o trabalho desenvolvido, pois demonstra que o motor de busca desenvolvido utilizando o OpenCL apresenta desempenho superior em relação aos comparados. Além disso, os experimentos realizados comprovam que a adoção da programação baseada em GPUs oferece vantagens em relação à programação Sequencial e ou baseada em *Threads*.

VI. Considerações Finais

Neste artigo apresenta-se uma proposta de implementação de um motor de busca para a carga inicial dos dados na memória de trabalho dos nós da rede do algoritmo RETE, visando especificamente motores de inferência baseado em regras para processamento de bases de conhecimento.

O principal objetivo do trabalho aqui relatado é verificar o ganho de desempenho com a paralelização do motor busca, visando também definir qual paradigma de programação paralela oferece maiores ganhos.

Foram então apresentados aqui o desempenho de três motores de busca: um implementado sob o paradigma de programação sequencial e dois sob o paralelo. Para a implementação do paradigma de programação paralela foram utilizadas a implementação de *Threads* de paralelização em CPU e o framework OpenCL para paralelização em GPU. A fim de possibilitar a aplicação da paralelização baseada em GPU foi desenvolvido um algoritmo que permite processar algebricamente a base de conhecimento, resultando em uma simplificação do tratamento dos dados, que antes eram manipulados simbolicamente. A este processo de transformação dos dados manipulados pelo sistema de busca deu-se o nome de vetorização da base de conhecimento.

Foram demonstrados e analisados os resultados obtidos em experimentos realizados sobre extensas bases de conhecimento artificiais geradas aleatoriamente, possibilitando assim comparar-se e verificar-se os resultados obtidos.

O trabalho aqui apresentado constitui-se em uma etapa de um projeto de pesquisa que tem como objetivo disponibilizar

um mecanismo de inferência genérico baseado em regras de produção para ser utilizado no framework semântico Jena. A partir dos resultados obtidos nos experimentos realizados verificou-se que o motor de busca paralelizado oferece ganhos de desempenho expressivos, os quais indicam a continuidade dos trabalhos, visando agora a integração deste motor de busca na atual máquina de inferência lógica baseada em regras genéricas do Jena.

REFERÊNCIAS

[1] A. Lenci, S. Montemagni, V. Pirrelli, e G. Venturi, "NLP-based ontology learning from legal texts. A case study.", In Proceedings of the LOAIT, 2007, p. 113–129.

[2] E. D. Liddy, E. Hovy, J. Lin, J. Prager, D. Radev, L. Vanderwende, e R. Weischedel, "Natural language processing", Encycl. Libr. Inf. Sci., vol. 2, 2003.

[3] L. Lesmo, "The rule-based parser of the NLP group of the University of Torino", Intell. Artif., vol. 2, no 4, p. 46–47, 2007.

[4] B. A. Onyshkevych e S. Nirenburg, "Lexicon, ontology, and text meaning", Lexical Semantics and Knowledge Representation, Springer, 1992, p. 289–303.

[5] R. van der Goot e G. van Noord, "ROB: Using semantic meaning to recognize paraphrases", In Proceedings of the SemEval, 2015.

[6] F. A. P. de Paiva, J. A. F. Costa, e C. R. M. Silva, "An Ontology-Based Recommender System Architecture for Semantic Searches in Vehicles Sales Portals", Hybrid Artificial Intelligence Systems, M. Polycarpou, A. C. P. L.

[7] F. de Carvalho, J.-S. Pan, M. Woźniak, H. Quintian, e E. Corchado, Orgs. Springer International Publishing, 2014, p. 537–548.

[8] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem", Artif. Intell., vol. 19, no 1, p. 17–37, 1982.

[9] J. Urbani, S. Kotoulas, J. Maassen, F. Van Harmelen, e H. Bal, "OWL reasoning with WebPIE: calculating the closure of 100 billion triples", The Semantic Web: Research and Applications, Springer, 2010, p. 213–227.

[10] J. Urbani, S. Kotoulas, E. Oren, e F. Van Harmelen, Scalable distributed reasoning using mapreduce. Springer, 2009.

[11] F. Maier, R. Mutharaju, e P. Hitzler, "Distributed reasoning with EL++ using MapReduce", 2010.

[12] C. Liu, G. Qi, H. Wang, e Y. Yu, "Reasoning with large scale ontologies in fuzzy pD* using MapReduce", Comput. Intell. Mag. IEEE, vol. 7, no 2, p. 54–66, 2012.

[13] Y. Kazakov, M. Krötzsch, e F. Simančík, "Concurrent Classification of EL Ontologies", The Semantic Web–ISWC 2011, Springer, 2011, p. 305–320.

[14] Y. Ren, J. Z. Pan, e K. Lee, "Parallel ABox Reasoning of EL Ontologies", The Semantic Web, Springer, 2012, p. 17–32.

[15] N. Heino e J. Z. Pan, "RDFS reasoning on massively parallel hardware", In Proceedings of the The Semantic Web–ISWC 2012, Springer, 2012, p. 133–148.

[16] Martin Peters, C. Brink, S. Sachweh, e A. Zündorf, "Rule-based Reasoning on Massively Parallel Hardware.", In Proceedings of the SSWS@ ISWC, 2013, p. 33–49.

[17] Y. Gu, B.-S. Lee, e W. Cai, "Evaluation of Java thread performance on two different multithreaded kernels", ACM SIGOPS Oper. Syst. Rev., vol. 33, no 1, p. 34–46, 1999.

[18] R. Soma e V. K. Prasanna, "Parallel inferencing for OWL knowledge bases", In Proceedings of the Parallel Processing, 2008. ICPP'08. 37th International Conference on, 2008, p. 75–82.

[19] "Apache Jena Home Site". [Online]. Available at: http://jena.apache.org/.

[20] D. Brickley e L. Miller, "FOAF vocabulary specification 0.98", Namespace Doc., vol. 9, 2012.

[21] "Executors (Java Platform SE 8)". Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html>. Acessado: jan. 2018.