

# Artigo pro Cuda Fuzzy

Edevaldo Braga dos Santos<sup>1</sup>, Giovane de Oliveira Torres<sup>1</sup>, Guilherme Pereira Paim<sup>1</sup>,  
Renan Zafalon da Silva<sup>1</sup>, Vitor Alano de Ataides<sup>1</sup>, Maurício Lima Pilla<sup>1</sup>

<sup>1</sup>Universidade Federal de Pelotas  
Pelotas, RS - Brasil

{edevaldo.santos,gdotorres,gppaim,renan.zafalon,vaataides,pilla}@inf.ufpel.edu.br

**Abstract.** *Abstract aqui.*

**Resumo.** *Resumo aqui.*

## 1. Introdução

Existem diversos casos onde classes de objetos não pertencem totalmente a um conjunto. Baseado nisso, Zadeh definiu a teoria dos conjuntos *fuzzy* [Zadeh 1965], que visa tratar problemas de imprecisão ao classificar dados no mundo real. Os conjuntos *fuzzy* possuem aplicações em sistemas de controle e de suporte a decisão, onde a descrição do problema não é feita de forma precisa [Weber and Klein 2003].

Utilizando os conhecimentos provenientes dos conjuntos *fuzzy*, se tem a base para a utilização da lógica *fuzzy*, sendo construída a partir da lógica proposicional. Com isso, os operadores foram definidos a partir dos já estabelecidos na lógica clássica, com a adição de outros, para fins práticos [Tanscheit 2004]. Uma característica interessante, que diferencia a lógica tradicional da *fuzzy*, é que na primeira os valores que utilizados atendem à condição de serem verdadeiros ou falsos (0 ou 1). Já na segunda, trabalha-se com conjuntos *fuzzy*, os quais podem assumir um valor que pertence ao intervalo  $[0, 1]$ . Isso permite que um conjunto *fuzzy* possa ser representado por uma infinidade de valores [Klir and Yuan 1995].

Com a finalidade de obter computação com bom desempenho, se torna importante o uso dos vários núcleos de processamento, disponibilizados nos sistemas de computação atuais, visando um melhor aproveitamento do paralelismo do sistema computacional. Neste contexto, encaixam-se as GPUs (*Graphical Processor Units*), que são componentes com alto poder de paralelismo [Sengupta et al. 2007]. Porém, é importante ressaltar que as GPUs são reservadas a obter bom desempenho com aplicações que possuem determinadas características [Owens et al. 2008], que incluem: (i) Requisitos computacionais excessivos, (ii) Paralelismo nas aplicações e (iii) maior importância do *throughput* em detrimento da latência. Destacam-se ainda alguns exemplos práticos, bem-sucedidos, que utilizam CUDA: Através do uso do poder computacional de CUDA, para análise do fluxo de tráfego aéreo, foi possível reduzir o tempo de análise do tráfego aéreo nacional de dez minutos para três segundos. Outro exemplo relevante é o ganho de desempenho em simulações moleculares NAMD (dinâmica molecular em nanoescala), nelas o ganho de desempenho foi possível graças às arquiteturas paralelas das GPUs [NVIDIA 2015].

Tendo sido discutidos estes conceitos, o objetivo deste trabalho é descrever uma biblioteca de lógica *fuzzy* voltada para GPUs, a fim de verificar a forma como pode ser efetuada uma implementação que consiga extrair paralelismo deste tipo de arquiteturas.

O restante deste artigo está dividido da seguinte maneira: A seção 2 fala sobre a lógica fuzzy, que é a base para a construção deste trabalho. Na 3, é descrita a implementação efetuada da biblioteca CudaFuzzy. A 4 destina-se a explicar a metodologia empregada para a execução de testes na biblioteca. Em 5 são exibidos e discutidos os principais resultados obtidos por este trabalho, de onde se tiram as principais conclusões, observadas na seção 6, a qual ainda mostra possíveis trabalhos futuros. Por fim, nos 7 faz-se uma breve discussão sobre os trabalhos que relacionam-se ao escopo deste artigo.

## 2. Lógica Fuzzy

A lógica *fuzzy* foi criada com base na teoria de conjuntos *fuzzy*. A principal ideia da lógica *fuzzy* é aumentar a representação de valores, não restringindo-se somente aos valores 0 (falso) e 1 (verdadeiro). Um valor na lógica *fuzzy* é representado por um número real que está entre os números 0 e 1 – e esse, representa o grau de pertinência do valor dentro de um conjunto *fuzzy* [Boclin and de Mello 2006]. A teoria dos conjuntos *fuzzy* foi criada com o propósito de oferecer ferramentas matemáticas que visam solucionar problemas que possuam algumas das seguintes características [Falcão 2002]: Incerteza, informações vagas e ambiguidade.

### 2.1. Operadores Fuzzy

Conjuntos são definidos por uma condição específica que define se um conjunto pertence ou não a outro. Para exemplificar os operadores em *fuzzy*, logo abaixo foram utilizados conjuntos A e B. Os mais comuns em conjuntos *fuzzy* são:

- União:  $(A \cup B = x \mid x \in A \vee x \in B)$
- Intersecção:  $(A \cap B = x \mid x \in A \wedge x \in B)$

Além dos operadores em conjuntos, os operadores aritméticos são fundamentais para a matemática *fuzzy*. Os operadores adição e subtração são mais fáceis de manipular, já os procedimentos de multiplicação e divisão são mais complexos. Novamente utilizando como exemplo dois conjuntos A e B, as principais operações aritméticas são:

- Adição -  $[a_1, a_3](+)[b_1, b_3] = [a_1 + b_1, a_3 + b_3]$
- Subtração -  $[a_1, a_3](-)[b_1, b_3] = [a_1 - b_1, a_3 - b_3]$
- Multiplicação -  $[a_1, a_3](\bullet)[b_1, b_3] = [a_1 \bullet b_1, a_3 \bullet b_3]$
- Divisão -  $[a_1, a_3](/)[b_1, b_3] = [a_1/b_1, a_3/b_3]$

## 3. CudaFuzzy

CudaFuzzy é o nome dado à biblioteca de lógica *fuzzy* desenvolvida neste trabalho, voltada a GPUs. Para o seu desenvolvimento, a biblioteca criada teve a necessidade de incluir vários operadores, para trabalhar sobre os conjuntos *fuzzy*. Foram escolhidas operações básicas [Klir and Yuan 1995] para a implementação, sendo estas: duas operações AND ( $\wedge$ ), duas operações OR ( $\vee$ ) e três operações NOT ( $\sim$ ). Na Figura 1 estão demonstradas as operações através dos operadores desenvolvidos.

Os operadores foram implementados em C, utilizando-se a biblioteca CUDA, e foram desenvolvidos como funções que recebem como argumentos um ou dois valores do tipo `double` (um para operações NOT, e dois para operações AND e OR), sendo esses os

$$\begin{aligned}
AND1(x, y) &= \min(x, y) \\
AND2(x, y) &= x \times y \\
OR1(x, y) &= \max(x, y) \\
OR2(x, y) &= (x + y) - (x \times y) \\
NOT1(x) &= 1 - x \\
NOT2(x) &= \sqrt{1 - x^2} \\
NOT3(x) &= \sqrt[3]{1 - x^3}
\end{aligned}$$

**Figura 1. Operações implementadas na biblioteca CudaFuzzy**

```

inline __device__ double d_Not3 (double x) {
    return pow(1 - pow(x, 3), 1.0 / 3);
}

```

**Figura 2. Código de implementação da operação de lógica fuzzy NOT3**

operandos e retornando um valor do tipo `double`, o qual é o resultado da operação. Na Figura 2 está exemplificada a implementação da operação NOT3.

Porém, é importante observar que a implementação destes operadores, por si só, representa pouco do trabalho efetuado. Como as arquiteturas alvo para esta biblioteca são as GPUs, houve a necessidade de elaborar uma maneira de aproveitar o paralelismo disponível nestas arquiteturas. A estratégia abordada por este trabalho buscou agrupar conjuntos de operações de um mesmo tipo, em um *array* de instruções. Com isso, surge a possibilidade de serem executadas múltiplas operações de lógica *fuzzy* nos vários núcleos disponibilizados pela GPU. As operações são agrupadas em um mesmo tipo para serem executadas em paralelo devido à taxonomia destas arquiteturas, sendo elas consideradas SIMT (*Single instruction multiple thread*) [Keckler et al. 2011], Através da arquitetura SIMT uma mesma intrução é transmitida para várias unidades de execução, essa arquitetura é muito utilizada de forma a acelerar os processamentos nos sistemas de computação em alto desempenho, pois os núcleos dos processadores são grandes e os processadores SIMT executam uma grande quantidade de threads com um mesmo fluxo de instrução utilizando diferentes dados. [Lai et al. ]

Assim, as operações são feitas sobre uma função que leva o prefixo `Bulk`, o qual fará uma operação de lógica *fuzzy* sobre um *array* de operandos (Em caso de operações `BulkNOT`) ou sobre dois *arrays* de operandos (Em caso de operações `BulkAND` e `BulkOR`). No caso de operações que trabalham com dois *arrays*, essas serão efetuadas de maneira correspondente nos *arrays*, i.e., o primeiro operando de um *array* será combinado com o primeiro operando do outro *array*, gerando assim uma computação parcial da operação, sendo esse o primeiro elemento do *array* de resultado. Logo, quando a operação requer dois *arrays* como entrada, os mesmos deverão conter o mesmo número de elementos. Na Figura 3 está representada uma computação parcial da operação `BulkAND1`.

O código acima é a representação de uma operação efetuada entre o *i*-ésimo elemento de um *array* com seu correspondente no segundo *array*. O resultado desta operação

```

        cudaFree ( d_result );
        cudaFree ( d_array );

        return result; // Return results
    }

    /* ___kernel_And___

```

**Figura 3. Código de implementação de uma única operação BulkAND1**

gera o  $i$ -ésimo elemento do *array*, o qual armazenará os resultados da operação Bulk. Ao final da execução de todas as operações que trabalham com cada elemento do *array*, tem-se todos os resultados parciais no *array* de resultado.

## 4. Metodologia

A fim de extrair os resultados deste artigo, foi proposta uma metodologia. Após ser feita a implementação da biblioteca CudaFuzzy, também foi construída uma versão sequencial da biblioteca a fim de comparar o ganho de desempenho paralelizando o código. Esta versão, em vez de executar em paralelo as operações dentro de um *array*, é efetuado o cálculo de uma operação por vez.

Foi feita então uma bateria de execuções para a geração dos resultados. Cada operação Bulk foi executada XX vezes, em sua versão paralelizada usando GPU, bem como a versão sequencial. Para extrair os tempos de execução e comparar as duas versões, obteve-se a média do total de execuções, além de efetuar o teste t de Student para comprovar a diferença estatística entre as médias. Por fim, foram feitas análises estatísticas quanto à variabilidade dos tempos de execução, fazendo uso de gráficos de dispersão e de caixa para tal fim.

Não sabemos

Será feito? Não sei

## 5. Resultados e Discussão

## 6. Conclusões e Trabalhos Futuros

## 7. Trabalhos Relacionados

Existem diversas implementações relacionadas a lógica *fuzzy*. Os artigos visualizados na bibliografia normalmente fazem utilização de lógica *fuzzy* voltada para um tipo de problema, não descrevendo uma biblioteca genérica. Como trabalhos descritos desta forma, pode-se citar [Sugeno and Yasukawa 1993], que utiliza lógica *fuzzy* para a discussão de um método para modelagem qualitativa, e [Li et al. 2011], que emprega a lógica *fuzzy* para fazer o gerenciamento de energia em baterias de automóveis híbridos do tipo *plug-in*.

No que diz respeito às bibliotecas relacionadas a lógica *fuzzy*, existe uma desenvolvida na linguagem Java, chamada de jFuzzyLogic [Cingolani and Alcalá-Fdez 2012, Cingolani and Alcalá-Fdez 2013]. Esta biblioteca é uma implementação de sistemas *fuzzy* que permite projetar controladores de lógica *fuzzy*. Por fim, existe a biblioteca FuzzyGPU [Defour and Marin 2014], que é o trabalho relacionado mais próximo ao que este artigo apresenta. FuzzyGPU é uma implementação de uma biblioteca de aritmética *fuzzy* voltada a GPUs.

Falta fazer um paralelo entre o que foi feito neste trabalho de diferente com os trabalhos relacionados

## Referências

- Boclin, A. d. S. C. and de Mello, R. (2006). A decision support method for environmental impact assessment using a fuzzy logic approach. *Ecological economics*, 58(1):170–181.
- Cingolani, P. and Alcalá-Fdez, J. (2012). jfuzzylogic: a robust and flexible fuzzy-logic inference system language implementation. In *FUZZ-IEEE*, pages 1–8. Citeseer.
- Cingolani, P. and Alcalá-Fdez, J. (2013). jfuzzylogic: a java library to design fuzzy logic controllers according to the standard for fuzzy control programming. *International Journal of Computational Intelligence Systems*, 6(1):61–75.
- Defour, D. and Marin, M. (2014). Fuzzygpu: a fuzzy arithmetic library for gpu. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pages 624–631. IEEE.
- Falcão, D. M. (2002). Conjuntos, lógica e sistemas fuzzy. *COPPE/UFRJ*.
- Keckler, S. W., Dally, W. J., Khailany, B., Garland, M., and Glasco, D. (2011). Gpus and the future of parallel computing. *IEEE Micro*, (5):7–17.
- Klir, G. and Yuan, B. (1995). *Fuzzy sets and fuzzy logic*, volume 4. Prentice Hall New Jersey.
- Lai, B.-C., Platero, L., and Kuo, H.-K. A quantitative method to data reuse patterns of simt applications.
- Li, S. G., Sharkh, S., Walsh, F. C., and Zhang, C.-N. (2011). Energy and battery management of a plug-in series hybrid electric vehicle using fuzzy logic. *Vehicular Technology, IEEE Transactions on*, 60(8):3571–3585.
- NVIDIA (2015). Plataforma de computação paralela | cuda | nvidia | nvidia. Disponível em: <[http://www.nvidia.com.br/object/cuda\\_home\\_new\\_br.html](http://www.nvidia.com.br/object/cuda_home_new_br.html)>.
- Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., and Phillips, J. C. (2008). Gpu computing. *Proceedings of the IEEE*, 96(5):879–899.
- Sengupta, S., Harris, M., Zhang, Y., and Owens, J. D. (2007). Scan primitives for gpu computing. In *Graphics hardware*, volume 2007, pages 97–106.
- Sugeno, M. and Yasukawa, T. (1993). A fuzzy-logic-based approach to qualitative modeling. *IEEE Transactions on fuzzy systems*, 1(1):7–31.
- Tanscheit, R. (2004). Sistemas fuzzy. *Inteligência computacional: aplicadaa administração, economia e engenharia em Matlab*, pages 229–264.
- Weber, L. and Klein, P. A. T. (2003). *Aplicação da lógica fuzzy em software e hardware*. Editora da ULBRA.
- Zadeh, L. A. (1965). Fuzzy sets. *Information and control*, 8(3):338–353.