# Além do fgssjoin: Algoritmos Paralelos para Junções por Similaridade de Conjuntos

**Rafael David Quirino, Wellington Santos Martins**

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Alameda Palmeiras, Quadra D, Campus Samambaia 131, Goiania - GO, Brazil,

rafaeldavid@inf.ufg.br, wsmartins@ufg.br

***Abstract.*** *Set similarity joins are very important operations on modern database systems, specially for the so called data warehouses, where several daily operations like integration, cleaning and data mining use it frequently. Exact algorithms, which yields all possible similar pairs regarding some similarity threshold are computationally expensive, which imposes slowness for analytical workloads and highlights the need for parallel solutions. Recent work shows parallel algorithms designed for devices with many-core architectures like GPUs. In this paper we present a new algorithm for the filtering phase of the fgssjoin, a well known parallel gpu-based algorithm for exact set similarity joins.*

***Resumo.*** *Junções por similaridade de conjuntos são operações de grande importância nos sistemas modernos de bancos de dados, especialmente para os chamados armazéns de dados, onde várias operações rotineiras como integração, limpeza e mineração de dados as utilizam com frequência. Algoritmos exatos, que retornam todos os pares similares possíveis de acordo com algum limiar de similaridade são computacionalmente caros, o que impõe lentidão a cargas de trabalho analíticas e destaca a necessidade de soluções paralelas para o problema. Trabalhos recentes apresentam algoritmos paralelos voltados para dispositivos de arquitetura many-core como as GPUs. Nesse artigo apresentamos um novo algoritmo para a etapa de filtragem do fgssjoin, um algoritmo paralelo conhecido, baseado em gpu, para a junção exata por similaridade de conjuntos.*

## 1. Introdução

As junções são operações ubíquas em sistemas de bancos de dados, principalmente nos relacionais, e a natureza quadrática dessas operações (imagine uma junção de uma relação consigo mesma) impõe muitos desafios a construção de algoritmos eficientes para sua execução. A operação de junção por similaridade retorna todos os pares de registros de uma base de dados (auto-junção) ou entre duas bases de dados, para os quais o resultado da aplicação de uma função de similaridade definida sobre o espaço de pares em questão não é menor do que um dado limiar. Quando falamos em junção por similaridade de conjuntos estamos nos referindo a bases de dados cujos registros (ou partes dos registros) podem ser representandos como conjuntos, coleções de elementos sem repetição. O problema vem atraindo atenção nos últimos anos [Sarawagi and Kirpal , Chaudhuri et al. 2006, Bayardo et al. 2007, Vernica et al. 2010, Xiao et al. 2011, Ribeiro and Härder 2011, Wang et al. 2012, Cruz et al. 2015], em

grande parte por conta do aumento no volume e na complexidade dos dados no contexto do *Big Data* mas também pelo fato de ser uma operação importante tanto por si só quanto como etapa intermediária em algoritmos mais complexos de processamento de dados, como integração de dados [Doan et al. 2012], limpeza de dados [Chaudhuri et al. 2006] e mineração de dados [Leskovec et al. 2014].

Dado esse contexto, fica evidente a necessidade de soluções paralelas capazes de explorar as modernas arquiteturas de múltiplos núcleos (*multi-core*) ou de muitos núcleos (*many-core*). As GPUs tem se mostrado bastante atrativas, pois oferecem a possibilidade de paralelismo massivo e otimizações focadas em vazão de dados e desempenho por Watt, o que se traduz em alto poder computacional e eficiência energética. Entretanto, por causa da diferença na arquitetura e no modelo de programação, a utilização eficiente desses dispositivos requer uma quantidade significativa de paralelismo e um esforço de desenvolvimento de algoritmos otimizados para esse ambiente.

## 2. Junção Paralela e o Algoritmo Fgssjoin

O algoritmo fgssjoin [Quirino et al. 2017] define um esquema de iteração que processa blocos de registros da base de dados de maneira independente, em ciclos de indexação-filtragem-verificação. As etapas de indexação e de filtragem utilizam apenas um subconjunto dos elementos dos registros da base de dados, chamado de prefixo, técnica conhecida como filtragem por prefixo.

### 2.1. Indexação

Nos algoritmos do estado-da-arte, as listas do índice invertido são criadas durante o processo de filtragem, o que os torna inerentemente algoritmos sequenciais: conjuntos são sequencialmente testados contra o índice e o estado das listas em uma iteração depende do seu estado na iteração anterior. Para resolver este problema, precisamos criar todo o índice invertido estaticamente antes da fase de filtragem. Desta forma, podemos realizar sondagens de forma independente, porque o índice está sempre completo. Portanto, precisamos de um algoritmo paralelo eficiente para criar o índice invertido.

### 2.2. Filtragem

Com todo o índice invertido armazenado na memória, podemos configurar cada processador para executar uma sondagem (processar um conjunto) em relação ao índice, aplicando as técnicas de filtragem discutidas em [Quirino et al. 2017], especialmente a filtragem por prefixo e por tamanho de conjunto.

### 2.3. Verificação

A fase de verificação, que finalmente produz o resultado final, pode ser processada trivialmente em paralelo. Ele consiste simplesmente em realizar o cálculo da intersecção de cada par candidato para verificar se é suficiente para qualificá-lo como um par similar. Isso pode ser feito facilmente em paralelo, fazendo com que cada processador execute a verificação em um par candidato.

### 2.4. Particionamento em blocos

A necessidade de vetores quadráticos na fase de filtragem estabelece um limite para o tamanho dos bancos de dados que podemos processar. Para resolver este problema, precisamos dividir nosso espaço de pesquisa em blocos que se enquadram nos requisitos de

memória. O algoritmo processa todas as combinações de pares de blocos onde o primeiro precede ou é igual ao segundo, sendo este o bloco de indexação e o primeiro o de consulta.

## 3. Novo algoritmo de filtragem

O algoritmo *fgssjoin* tem uma granularidade de uma *thread* por conjunto durante a filtragem. Para cada par de blocos é inicializada uma matriz de intersecções parciais computadas durante a filtragem, e que é compactada para gerar os pares candidatos. A figura 2 ilustra o novo algoritmo de filtragem proposto. Nele a granularidade é de um bloco de threads por conjunto. Em cada bloco definimos uma grade de memória compartilhada que ira varrer as listas invertidas e executar uma redução, a cada passo, para acumular as intersecções parciais. O resultado então pode ser enviado para a memória global de maneira aglutinada. O processo de carregamento das listas invertidas para a memória compartilhada permite descartar a inicialização de uma matriz de intersecções parciais e o processo de redução possibilita descartar a compactação dos pares candidatos. Ambos os processos representam gargalos do algoritmo clássico, principalmente para limiares altos de similaridade.

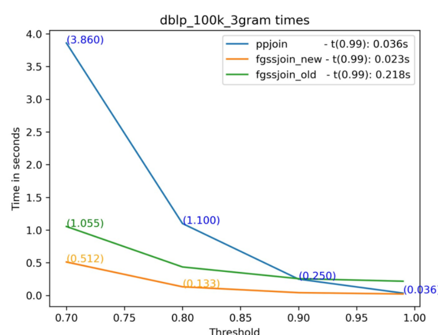


**Figura 1. Resultados dos experimentos.**

Experimentos foram realizados com um arquivo de 100 mil registros da base DBLP, em uma máquina com um processador i7 8750H, 32GB de memória RAM e uma GPU NVIDIA GTX 1050ti, de 4GB de memória e 768 núcleos CUDA. Foram comparados 3 algoritmos, o ppjoin, algoritmo sequencial de filtragem por prefixos do estado da arte, a versão original do fgssjoin, e uma nova versão com o algoritmo de filtragem apresentado neste trabalho. Os resultados mostram que o novo algoritmo foi mais rápido que a versão original, e mais rápido que o algoritmo sequencial mesmo em limiares muito altos de similaridade.

## 4. Conclusões e Trabalhos Futuros

Apesar dos bons resultados apresentados pelo trabalho do algoritmo fgssjoin, ainda há espaço para melhorias, principalmente em relação à etapa de filtragem, que utiliza estruturas de dados de ordem quadrática em relação ao tamanho dos blocos, e não utiliza a memória compartilhada da GPU. Nesse trabalho oferecemos um novo algoritmo para essa etapa que resolve esses problemas. A ideia é fazer com que cada multiprocessador da GPU processe um conjunto de consulta, carregando os dados das listas invertidas do prefixo desse conjunto para a memória compartilhada, melhorando o padrão de acesso

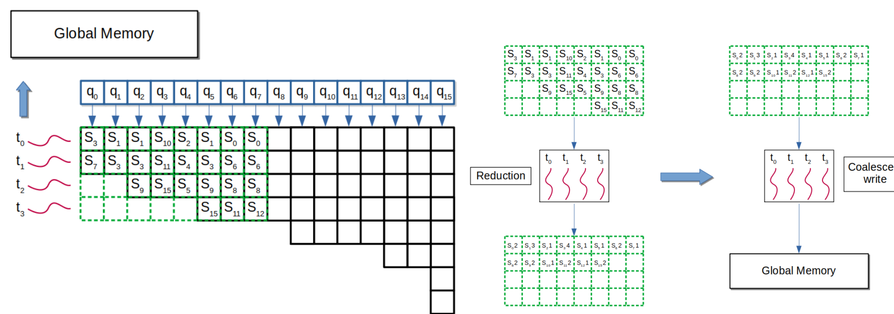


**Figura 2. Ilustração do processo de carregamento das listas invertidas para a memória compartilhada e da redução que gera os pares candidatos.**

e a reutilização da memória, e dispensando a necessidade de compactação da matrix de intersecções parciais bem como de reconfiguração da memória a cada ciclo de filtragem-verificação, que, de acordo com os experimentos, representam gargalos para limiares altos de similaridade.

## Referências

Bayardo, R. J., Ma, Y., and Srikant, R. (2007). Scaling up All Pairs Similarity Search. In *WWW*, pages 131–140.

Chaudhuri, S., Ganti, V., and Kaushik, R. (2006). A primitive operator for similarity joins in data cleaning. In *ICDE*, page 5.

Cruz, M. S. H., Kozawa, Y., Amagasa, T., and Kitagawa, H. (2015). GPU Acceleration of Set Similarity Joins. In *DEXA*, pages 384–398.

Doan, A., Halevy, A. Y., and Ives, Z. G. (2012). *Principles of Data Integration*. Morgan Kaufmann.

Leskovec, J., Rajaraman, A., and Ullman, J. D. (2014). *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press.

Quirino, R. D., Ribeiro-Júnior, S., Ribeiro, L. A., and Martins, W. S. (2017). fgssjoin: A gpu-based algorithm for set similarity joins. In Hammoudi, S., Smialek, M., Camp, O., and Filipe, J., editors, *ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems, Volume 1, Porto, Portugal, April 26-29, 2017*, pages 152–161. SciTePress.

Ribeiro, L. A. and Härder, T. (2011). Generalizing Prefix Filtering to Improve Set Similarity Joins. *Information Systems*, 36(1):62–78.

Sarawagi, S. and Kirpal, A. Efficient Set Joins on Similarity Predicates. In *SIGMOD*.

Vernica, R., Carey, M. J., and Li, C. (2010). Efficient Parallel Set-similarity Joins using MapReduce. In *SIGMOD*, pages 495–506.

Wang, J., Li, G., and Feng, J. (2012). Can We Beat the Prefix Filtering?: An Adaptive Framework for Similarity Join and Search. In *SIGMOD*, pages 85–96.

Xiao, C., Wang, W., Lin, X., Yu, J. X., and Wang, G. (2011). Efficient Similarity Joins for Near-duplicate Detection. *TODS*, 36(3):15.