Decidindo entre GPU e CPU para Processar Grafos com Base em Métricas de Alto Nível

Marcelo K. Moori[1], **Hiago Mayk G. de A. Rocha**[1], **Janaina Schwarzrock**[1], **Arthur F. Lorenzon**[2] **e Antonio Carlos S. Beck**[1]

[1]Universidade Federal do Rio Grande do Sul – Porto Alegre – RS – Brasil

[2]Universidade Federal do Pampa – Alegrete – RS – Brasil

mkmoori@inf.ufrgs.br

Resumo. *Apesar dos avanços nas GPUs modernas ter acelerado a execução de aplicações que processam grandes quantidades de dados, acelerar o processamento de grafos nesses sistemas não é uma tarefa trivial: aplicações de grafos são caracterizadas por alto volume de acesso irregular à memória e que varia com a estrutura dos grafos, fazendo com que muitas vezes elas não alcancem seus picos de performance quando executadas em GPUs. Nesses casos, a execução em CPU é mais adequada. Felizmente, as estruturas dos grafos podem ser identificadas por meio de métricas de alto nível (e.g., diâmetro e coeficiente médio de clusterização), e elas podem auxiliar o projetista na tomada de decisão de onde executar a aplicação, se em GPU ou em CPU. Neste trabalho, nós propomos uma metodologia que usa essas características de alto nível em uma Regressão Linear para auxiliar na tomada de decisão de onde processar os grafos, em GPU ou CPU. Assim, sempre que um novo grafo precisar ser processado, a decisão de onde processá-lo será tomada com base nessas métricas, evitando qualquer execução adicional dos algoritmos de grafos. Nosso resultados experimentais, considerando 1 GPU e 2 CPUs, mostram que as métricas mais relevantes dos grafos variam conforme os algoritmo e as máquinas onde esses grafos serão executados. Nossa proposta apresentou uma acurácia média de 85% quando aplicado uma Regressão Linear com as características mais relevantes.*

1. Introdução

Grafos são estruturas de dados poderosas capazes de modelar problemas de diversos domínios, tais como na química, física, neurociência, linguística computacional, análises de redes sociais, dentre muitas outros. Devido a sua vasta aplicabilidade, desenvolver métodos eficientes para a exploração e descoberta de conhecimento em massivas quantidades de dados estruturados em grafos vem sendo uma área de grande interesse [Rocha et al. 2021, Moori et al. 2021]. Contudo, o rápido aumento no tamanho dos grafos extraídos de fontes reais (e.g., atualmente, o grafo da *World Wide Web* contém mais de 3,72 bilhões de páginas[1]) faz com que o uso de novas tecnologias seja fundamental para dar suporte ao seu processamento. Nesse contexto, os avanços no desenvolvimento das *Graphic Processing Units* (GPUs), sobretudo no aspecto de possibilitar paralelismo massivo e alta largura de banda, têm atraído a atenção dos pesquisadores para acelerar a computação de algoritmos de grafos.

As GPUs são arquiteturas SIMT (*Single Instruction, Multiple Threads*), onde os ganhos de desempenho são dados através da exploração do paralelismo massivo

[1]*The size of the World Wide Web*: https://www.worldwidewebsize.com/. Acessado em 12/07/2022.

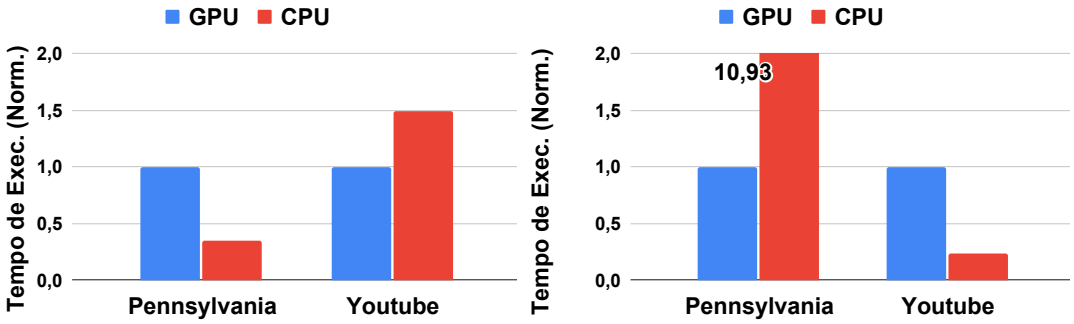
de dados. Grande parte da área da GPU é usada por unidades simples de processamento, enquanto uma fração menor é responsável pelo controle e caches. Com isso, as GPUs são excelentes para acelerar computações regulares que têm disponível um alto grau de paralelismo. Por outro lado, os processadores multicore com vários núcleos do tipo superescalar exploram o paralelismo no nível de instruções, *threads* e dados (nestes, porém, em uma escala bem menor que as GPUs, através de instruções SIMD - *Single Instruction Multiple Data*). Comparado com as GPUs, as CPUs - que possuem unidades de controle complexas e grandes *caches* [Khorasani et al. 2014] - geralmente são melhores em realizar tarefas que demandam baixa latência, e que não possuem níveis de paralelismo de dados amplamente disponíveis para serem explorados [Lorenzon et al. 2017, Lorenzon and Beck Filho 2019].

Apesar das GPUs oferecerem um alto grau de paralelismo, usá-las para acelerar a computação de grafos pode não ser uma tarefa trivial. As aplicações de grafos apresentam padrão irregular de acessos à memória, fazendo com que elas possam não alcançar seus picos de desempenho quando executadas em GPUs. Além disso, devido ao fato do tamanho de memória da GPU ser limitada (considerando o tamanho massivo de dados para processamento em grafos), mover dados da memória da CPU para a memória da GPU também pode ser um fator limitante. Ainda, os desvios condicionais (e.g., as instruções *if-else*, o que causa divergência de desvios) nas computações de grafos fazem com que o alto grau de paralelismo oferecido pelas execuções da GPU não sejam completamente explorados, que pode degradar consideravelmente o desempenho das aplicações [Hong et al. 2011].

Os *frameworks* de processamento de grafos atuais, seja para GPU (e.g., GraphBlast [Yang et al. 2022], SuiteSparse [Davis 2019] e CuSha [Khorasani et al. 2014]) ou CPU (e.g., Ligra [Shun and Blelloch 2013], e GraphGrind [Sun et al. 2017]), implementam algum tipo de particionamento no grafo ou alteram a forma como o grafo é avaliado durante a execução dos algoritmos com o objetivo de melhorar os seus tempos de execução. Contudo, eles estão longe de encontrar soluções ótimas para muitos grafos devido a grande quantidade de dados e as diferentes estruturas que eles podem ter (e.g., malhas e redes sociais). O motivo é que o desempenho das aplicações de grafos não depende apenas do algoritmo, ou da estratégia de otimização aplicada sobre ele, mas também da estrutura topológica do grafo sendo processado [de A. Rocha et al. 2022].

Entretanto, essa diferença na estrutura poderia indicar se um grafo é melhor processado em GPU ou em CPU. Para ilustrar a discussão acima, considere dois grafos com topologias distintas (ver Tabela 1 na seção 2: *Pennsylvania* com Diâmetro (Diâ.) 786 e Coeficiente de Médio Clusterização (CMC) 4,6e-2, representando uma malha; e o *Youtube* com Diâ. 20 e CMC 8,0e-2, representando uma rede social. A Figura 1 apresenta os tempos de execução (eixo-y) dos algoritmos *Breadth-First Search* (BFS) e o *Triangle Couting* (TC) quando executando duas entradas diferentes (*Pennsylvania* e *Youtube*) em GPU e CPU (barras com diferentes cores no eixo x). Os resultados são normalizados pela execução em GPU, logo, quanto menor, melhor em CPU (menor tempo de execução). Como observado na Figura 1, enquanto o BFS-*Pennsylvania* tem melhor desempenho quando executado em CPU (2,86x), o BFS-*Youtube* tem sua melhor desempenho executado em GPU (1,48x). O mesmo acontece com o TC, onde o TC-*Pennsylvania* é melhor em GPU (10,94x) e o TC-*Youtube* é melhor em CPU. Note ainda que existe variação quando se fixa o grafo e muda o algoritmo (e.g., enquanto o BFS-*Pennsylvania* é melhor em CPU, o TC-*Pennsylvania* tem sua melhor execução em GPU).

Com base nisso, este trabalho parte da premissa que a estrutura dos grafos impacta nas execuções dos algoritmos e, portanto, o dados que descrevem esta estrutura, tais como número de vértices e arestas, diâmetro e o coeficiente de clusterização (ver outras na Tabela 1), podem ser de alguma maneira usados para tomadas de decisão [de A. Rocha et al. 2022] - no caso, para decidir se é melhor executar determinado grafo



(a) BFS (b) TC

Figura 1. Comparação da execução em GPU e CPU dos algoritmos BFS e TC executando os grafos *Pennsylvania* e *Youtube*. Os experimentos foram executados no *Intel Xeon E5-2640 v2* (CPU) e na *NVIDIA Tesla K20m* (GPU).

em GPU ou CPU. Nota-se que essas características já são comumente disponíveis junto com os grafos [Leskovec and Krevl 2014]. Assim, elas podem ser exploradas antes da execução dos algoritmos, sem a necessidade de executar a aplicação de grafo para extrair suas características (isto é, nenhum *profiling* anterior à execução é necessário por parte do usuário).

Desta maneira, nós propomos uma metodologia para tomada de decisão na execução de algoritmos de grafos em GPU ou em CPU. Para isso, nossa metodologia consiste em: *(i)* correlacionar as características de alto nível de diferentes grafos com a execução de diferentes algoritmos em GPUs e CPUs; *(ii)* usar essas características para treinar uma Regressão Linear; e *(iii)* quando um novo grafo precisar ser processado, usar a Regressão Linear treinada com as características de alto nível do grafo já disponíveis para indicar se é melhor executar em GPU ou em CPU.

Em resumo, as principais contribuições deste artigo são:

- Uma metodologia que usa uma Regressão Linear para decidir se é melhor processar um determinado grafo em GPU ou em CPU; Nossa proposta se baseia apenas nas características de alto nível dos grafo, evitando qualquer execução adicional dos algoritmos que irão processá-los;
- Uma avaliação das execuções das aplicações de grafos em uma GPU e diferentes CPUs, destacando quais características que mais impactam na performance;
- Nosso experimentos, considerando 2 CPUs e 1 GPU, mostraram que as características relevantes variam conforme os algoritmos e as máquinas que processarão os grafos. Além disso, nossa proposta mostrou uma acurácia média de 85% quando aplicando a Regressão Linear com uma única característica dos grafos.

O restante deste artigo está organizado como segue: A seção 2 apresenta os conceitos básicos e as características dos algoritmos e grafos avaliados neste trabalho; Os detalhes da metodologia proposta são apresentados na seção 3; A metodologia experimental considerada nesse trabalho é apresentada na seção 4; A seção 5 apresenta os resultados; Os trabalhos relacionados são apresentados na seção 6. Por fim, a seção 7 conclui este trabalho.

2. Background

2.1. Grafos

Um grafo é uma estrutura matemática que consiste em um conjunto de vértices juntamente com um conjunto de arestas que conectam pares de vértices. Os grafos extraídos de fontes reais são comumente formados por uma quantidade massiva de dados e variam bastante

Tabela 1. Características dos grafos.

	Amazon	California	Google	Pennsyl.	Youtube	Berk	Texas	Wikitalk	Orkut	Patent
#Vértices	3,3e5	1,9e6	8,7e5	1,0e6	1,1e6	6,8e5	1,3e6	2,3e6	3,0e6	3,7e6
#Arestas	9,2e5	2,7e6	5,3e6	1,5e6	2,9e6	7,6e6	1,9e6	5,0e6	1,1e8	1,6e7
#Vért. no maior WCC	1	0,996	0,977	1	1	0,956	0,979	0,998	1	0,997
#Ares. no maior WCC	1	0,998	0,993	1	1	0,987	0,978	0,999	1	1
#Vért. no maior SCC	1	0,996	0,497	1	1	0,489	0,979	0,047	1	0
#Ares. no maior SCC	1	0,998	0,67	1	1	0,595	0,978	0,297	1	0
CMC	3,9e-1	4,6e-2	5,1e-1	4,6e-2	8,0e-2	5,9e-1	4,7e-2	5,2e-2	1,6e-1	7,5e-2
NT	6,6e5	1,2e5	1,3e7	6,7e4	3,0e6	6,4e7	8,2e4	9,2e6	6,2e8	7,5e6
FTF	7,9e-2	2,0e-2	1,9e-2	2,0e-2	2,0e-3	2,7e-3	2,0e-2	1,1e-3	1,4e-2	2,0e-2
Diâmetro	44	849	21	786	20	514	1054	9	9	22
90-PDE	15	500	8,1	530	6,5	9,9	670	4	4,8	9,4

em suas topologias, as quais podem ser caracterizadas pelas seguintes métricas de alto nível (ver Tabela 1): ***#Vértices*****:** número de vértices do grafo; ***#Arestas*****:** número de arestas do grafo; ***#Vert. no maior WCC*****:** o maior número de vértices no *Weakly Connected Components* (WCC). Ou seja, é um subconjunto maximal dos vértices do grafo com a restrição de que para quaisquer pares de vértices deve existir um caminho entre os vértices do par, ignorando a sua direção; ***#Ares. no maior WCC*****:** é o maior número de arestas no WCC. Ou seja, é um subconjunto maximal das arestas do grafo com a restrição de que para quaisquer pares de vértices deve existir um caminho entre os vértices do par, ignorando a sua direção; ***#Vert. no maior SCC*****:** é o maior número de vértices no *Strongly Connected Components* (SCC). Ou seja, é um subconjunto maximal dos vértices do grafo tal qual o WCC, mas com sua direção importando nesse caso (e.g., para um grafo não direcionado o SCC e WCC são equivalentes); ***#Ares. no maior SCC*****:** é o maior número de arestas no SCC. Ou seja, é um subconjunto maximal das arestas do grafo tal qual o WCC, mas com sua direção importando nesse caso (e.g., para um grafo não direcionado o SCC e WCC são equivalentes); ***CMC*****:** é o *Coeficiente Médio de Clusterização*. Ou seja, é a média dos coeficientes de agrupamentos locais de cada vértice do grafo, o que é definido pela proporção de ligações entre os vértices da sua vizinhança e o número de ligações que poderiam existir entre estes; ***NT*****:** É o Número de Triângulos no grafo, que representa o número de triplas (combinação de três vértices) conectados, ignorando a sua direção; ***FTF*****:** É a Fração de Triângulos Fechados no grafo. Ou seja, é a razão entre o NT e o total de triplas (todas as combinações de três vértices conectadas ou não); ***Diâmetro*****:** o mais longo menor caminho entre qualquer par de vértices do grafo; ***90-PDE*****:** É o 90-Percentil de Diâmetro Efetivo, representando a menor distância para qual 90% dos pares de uma amostra de 1000 vértices estão distanciados no máximo por esse valor.

2.2. Algoritmos para Processamento de Grafos

Vários trabalhos na literatura propõem *frameworks* que otimizam a execução paralela de algoritmos de processamento de grafos tanto em ambiente GPU [Yang et al. 2022, Davis 2019] quanto em CPU [Shun and Blelloch 2013, Mofrad et al. 2020]. Esses trabalhos propõem diferentes implementações de alto desempenho de algoritmos de amplamente utilizados pela indústria, como por exemplo o *PageRank* (PR) e o *Breadth-First Search* (BFS), com aplicabilidade em diversas áreas (e.g., química, medicina, comércio e logística).

A seguir, listamos alguns dos algoritmos mais utiliza na literatura de grafos e que a maioria dos *frameworks* implementam: ***Betweenness Centrality (BC)*** aproxima a medida de intermediação do grafo por um subconjuntos de vértices, tendo que a intermediação de um vértice v é dada pela razão entre os caminhos mínimos que atravessam v e todos os caminhamentos mínimos do conjunto considerado; ***Breadth-First Search (BFS)*** realiza a travessia do grafo a partir de um vértice raiz, visitando todos os vértices de mesmo nível antes de avançar para o próximo; ***PageRank (PR)*** iterativamente calcula o valor de

PageRank para todos os vértices com um fator de amortização d de 0,85; ***Single-Source Shortest Paths (SSSP)*** calcula todos os caminhamentos mínimos alcançáveis a partir de um dado vértice raiz; ***Triangle Counting (TC)*** computa o total de triângulos do grafo.

3. Proposta

A metodologia proposta, que auxilia o projetista na tomada de decisão na execução de aplicações de grafos em GPU ou CPU, avalia as características de diversos grafos e suas execuções em diferentes algoritmos e máquinas GPU/CPU. Nossa proposta usa regressão linear para decidir sobre qual é o melhor sistema (GPU ou CPU) para executar determinada aplicação de grafo avaliando apenas as características de alto nível do grafo a ser processado, como mostrado na seção 5. A Figura 2 apresenta o fluxo de execução da nossa metodologia, a qual funciona da seguinte forma:

Entradas. Neste passo, nossa metodologia recebe como entradas: *(i)* um conjunto de grafos a serem processados; *(ii)* um conjunto de arquivos executáveis dos algoritmos que processarão os grafos; *(iii)* a descrição das máquinas GPU e CPU disponíveis para executar as aplicações;

Pré-processamento. Dada as entradas, neste passo são extraídas as características de alto nível e executadas todas as combinações dos grafos, algoritmos e máquinas; essas características podem ser conhecidas de antemão ou extraídas pela ferramenta NetworKit [Staudt et al. 2016];

Análise das Características. Este passo consiste: no cálculo dos *speedups* considerando os tempos de execução na GPU e CPU; na análise das correlações entre as características dos grafos e os *speedups* calculados; e na indicação das características relevantes que podem ser usadas para a tomada de decisão;

Aplicação na Técnica de Predição. Neste passo, as características relevantes são aplicadas em uma Regressão Linear para a tomada de decisão na execução das aplicações de grafos em GPU ou em CPU (ver nosso estudo de caso na seção 5).

Nós automatizamos a metodologia proposta através de um *script* na linguagem Python versão 3.8.10. Para executar o *script* em qualquer sistema operacional Linux basta executar o comando: "python3 script.py input_grafos app_grafos desc_maquinas", onde *input_grafos*, *app_grafos*, e *desc_maquinas* se refere ao conjunto de grafos, conjunto de arquivos binários das aplicações e a descrição das máquinas disponíveis, respectivamente. Nossa metodologia trabalha com aplicações de grafos paralelizadas com as interfaces de programação paralela mais usadas, tais como OpenMP e POSIX Threads para execução em CPU, e CUDA e OpenGL para execução em GPU. Além disso, para obter os tempos de execução das aplicações, nossa metodologia usa a função time.time() do módulo **time** do Python3.

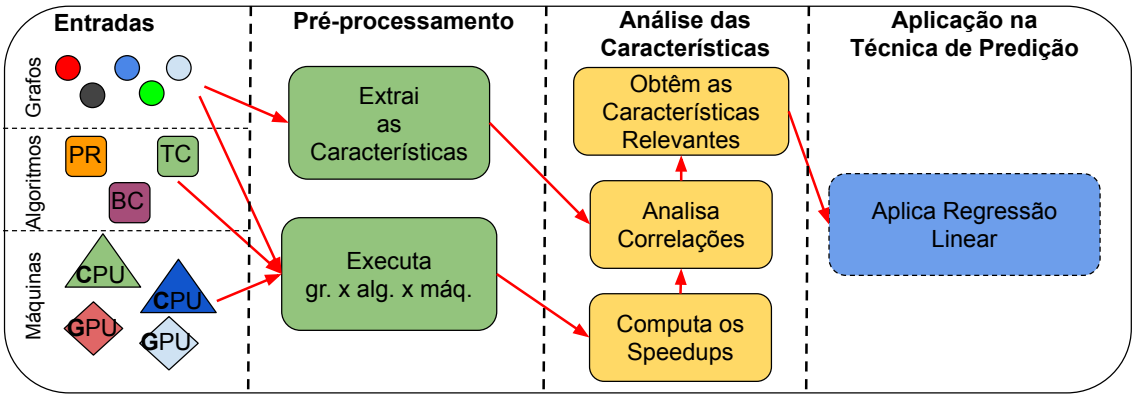


Figura 2. Visão geral da metodologia proposta.

Nas subseções seguintes, nós detalhamos cada um dos passos da metodologia proposta.

3.1. Entradas

Para a execução da metodologia proposta, as seguintes entradas são requeridas: ***Grafos.*** Um conjunto de grafos a serem processados: todos os grafos devem ser lidos antes de seu processamento. Nossa metodologia é capaz de trabalhar com os formatos mais usados na literatura, tais como lista de arestas de arestas (.el) e *Matrix Market format* (.mtx). ***Algoritmos.*** Um conjunto com os arquivos binários dos algoritmos de grafos em versões para GPU e CPU: todos eles devem ser compilados nos sistemas alvo antes de sua execução; ***Máquinas.*** A descrição das máquinas GPU e CPU disponíveis no sistema: elas devem estar disponíveis para uso, uma vez que a metodologia executará as aplicações para analisar os tempos de execução nos diferente sistemas (neste trabalho consideramos 3 diferentes máquinas, sendo 1 GPU e 2 CPUs – ver seção 4).

3.2. Pré-processamento

Nesse passo, *(i)* é feita a extração das características de alto nível dos grafos e *(ii)* o processamento dos grafos por todos os algoritmos executando nas diferentes máquinas. Para o primeiro caso *(i)*, as características dos grafos (e.g., diâmetro e coeficiente médio de clusterização) foram extraídas usando a ferramenta *NetworKit*, uma biblioteca paralela para a análise em larga escala de grafos [Staudt et al. 2016]. Contudo, se já se sabe essas características previamente, elas podem ser inseridas diretamente na metodologia, evitando ter que executar a ferramenta *NetworKit* para extraí-las. Para o segundo caso *(ii)*, são executadas todas as combinações de grafos, algoritmos e máquinas GPUs e CPUs com o objetivo de analisar suas performances nos passos seguintes.

3.3. Análise das Características

Nesse passo, *(i)* os *speedups* das execuções em GPU e CPU são analisados para cada grafo e algoritmo, *(ii)* são analisadas as correlações entre as características extraídas e os *speedups* computados, e *(iii)* são extraídas as características mais relevantes para serem aplicadas na técnica de predição. Em *(i)*, para tornar a análise genérica para qualquer número e tipo de GPU e CPU, é considerado o *speedup* das médias geométricas, dada pela seguinte formula: $\frac{GMEAN(gpu_1,gpu_2,...,gpu_n)}{GMEAN(cpu_1,cpu_2,...,cpu_m)}$, onde cpu_i é o tempo de execução na CPU $i \in [1,\ldots,n]$ e gpu_j é o tempo de execução na GPU $j \in [1,\ldots,m]$. Em *(ii)*, são computadas as matrizes de correlação levando em consideração os *speedups* computados e as características de alto nível dos grafos. Para isso, é usado o coeficiente de correlação de Pearson [Benesty et al. 2009], o qual mede o grau de correlação ρ entre duas variáveis, assim como a direção dessa correlação (se positiva ou negativa). Logo, o valor de ρ varia entre $[-1, 1]$ e quando $\rho = 1$ significa que a correlação é perfeita positiva (e.i., se uma aumenta, a outra sempre aumenta), quando $\rho = -1$ significa que a correlação é perfeita negativa (e.i., se uma aumenta, a outra sempre diminui), e quando $\rho = 0$ significa que não existe correlação entre as duas variáveis. Por fim, com base nos valores dos coeficientes ρ as características mais relevantes dos grafos são selecionadas para que possam ser usadas para decidir onde que vale a pena computar o grafo, se em GPU ou em CPU. Para isso, nós consideramos um limiar de $|\rho| = 0,5$, para que uma determinada característica seja considerada relevante.

3.4. Aplicação na Técnica de Predição

Por fim, as características relevantes extraídas são usadas para treinar uma Regressão Linear Simples. A Regressão Linear treinada será usada para tomada de decisão toda vez que um novo grafo precisar ser processado, fazendo uso apenas das suas características de alto nível e evitando qualquer execução adicional dos algoritmos de processamento de grafos.

Tabela 2. Configurações das máquinas usadas.

	CPU32	CPU128	GPU2496
Processador	Intel Xeon E5-2640v2	AMD Ryzen 3990X	NVIDIA Tesla K20m
Frequência base	2,0GHz	2,9GHz	706 MHz
#Núcleos / *#Threads*	16 / 32	64 / 128	2496
Cache L1	32 KB	32 KB	–
Cache L2	256 KB	512 KB	–
Cache L3	20 MB	16 MB	–
Memória RAM	128 GB	32 GB	5 GB

4. Metodologia

Algoritmos. Neste trabalho nós avaliamos os algoritmos de grafos disponíveis no GAP *Benchmark Suite* (GAPBS) [Beamer et al. 2015] para as execuções em CPU e os disponíveis no Gunrock [Wang et al. 2016] para execução em GPU. Ambos GAPBS e Gunrock disponibilizam implementações de alto desempenho de um conjunto representativo de algoritmos de processamento de grafos, sendo um *baseline* representativo do estado da arte para as pesquisas nessa área. Nós descrevemos todos os algoritmos avaliados na seção 2. Os algoritmo foram: *Betweenness Centrality* (BC), Breadth-First Search (BFS), *PageRank* (PR), *Single-Source Shortest Paths* (SSSP) e o Triangle Couting (TC). Todos eles são algoritmos tomados como base para a construção de aplicações mais complexas. Por exemplo, o BFS é usado no sistema de recomendação do Website Alibaba [Sahu et al. 2020]. Todos eles são implementados em C++11, paralelizados usando OpenMP (GAPBS) e CUDA (Gunrock) e suas implementações consideram a estratégia de otimização que é mais adequada para cada um deles. Em nossos experimentos, nós compilamos as aplicações usando GNU g++ 10.1.0 com a *flag* de otimização -O3. Também usamos OpenMP versão 4.5 nas aplicações do GAPBs e CUDA 11.4 nas aplicações do Gunrock.

Grafos. Nós avaliamos 10 grafos extraídos da SNAP [Leskovec and Krevl 2014]. Eles são descritos na seção 2 (ver Tabela 1). Nós consideramos um conjunto representativo de grafos do mundo real cobrindo as duas grandes classes de topologias (malhas e redes sociais) com diferentes características e tamanho (variando de 300 mil até 4 milhões de nós).

Ambiente de Execução. Nós realizamos os experimentos nos sistemas *multicore* apresentados na Tabela 2 usando o Sistema Operacional (SO) Linux com *kernel* v. 4.19.

5. Resultados Experimentais

Nessa seção, nós apresentamos os seguintes resultados: *(i)* uma análise das correlações das características de diversos grafos com o tempo necessários para o seu processamento em diferentes algoritmos; *(ii)* uma comparação das performances das aplicações de grafos quando executadas em GPU e em CPU; *(iii)* uma análise das características que impactam na performance quando as aplicações de grafos são executadas em GPU e em CPU; *(iv)* a execução completa da nossa metodologia, onde nós usamos as características identificadas na análise anterior em uma Regressão Linear.

(i) Correlações Características vs Máquinas. Na Tabela 3, nós apresentamos as correlações das características dos grafos com os tempos de execução quando executados por diferentes algoritmos (BFS, SSSP, BC, PR e TC) nas 3 máquinas avaliadas (CPU32, CPU128 e GPU2496). Coeficientes de correlação ρ maiores que 0,5 sugerem uma correlação moderada (menores que -0,5 indicam uma correlação no sentido contrário) e, por esse motivo, estão destacados em **negrito**. Valores maiores que 0,7 (ou menores que -0,7), por sua vez, sugerem uma alta correlação.

Tabela 3. Correlação entre os tempos de execuções e as métricas dos grafos.

	CPU32					CPU128					GPU2496				
	BFS	**SSSP**	**BC**	**PR**	**TC**	**BFS**	**SSSP**	**BC**	**PR**	**TC**	**BFS**	**SSSP**	**BC**	**PR**	**TC**
#Vértices	0,16	-0,12	0,34	**0,56**	0,36	-0,13	-0,13	-0,03	**0,73**	0,42	0,28	0,31	0,26	**0,55**	0,45
#Arestas	0,30	-0,25	**0,52**	**1,00**	**0,96**	-0,04	-0,26	0,05	**0,89**	**0,99**	**0,80**	**0,83**	**0,75**	**1,00**	**0,99**
#Vér. M. WCC	-0,17	-0,03	0,07	0,21	-0,02	-0,30	-0,04	-0,21	0,13	0,09	0,04	0,09	0,06	0,23	0,23
#Ares. M. WCC	-0,47	-0,46	-0,19	0,22	0,09	**-0,56**	-0,47	**-0,50**	0,19	0,15	-0,14	-0,08	-0,14	0,23	0,22
#Vér. m SCC	0,42	0,48	0,26	0,14	0,18	0,46	0,48	0,41	-0,11	0,20	0,49	0,48	**0,52**	0,17	0,25
#Ares. m. SCC	0,40	0,45	0,29	0,11	0,17	0,45	0,45	0,42	-0,16	0,19	0,47	0,46	**0,50**	0,12	0,23
CMC	-0,31	-0,47	-0,40	-0,07	0,14	-0,23	-0,46	-0,32	-0,06	0,05	-0,19	-0,20	-0,23	-0,08	-0,06
NT	0,32	-0,23	**0,55**	**0,98**	**0,98**	0,01	-0,24	0,10	**0,84**	**1,00**	**0,83**	**0,85**	**0,78**	**0,98**	**0,99**
FTF	-0,23	-0,01	-0,34	-0,13	-0,19	-0,17	-0,01	-0,23	-0,18	-0,15	-0,16	-0,16	-0,17	-0,11	-0,11
Diâ.	**0,79**	**0,95**	0,46	-0,29	-0,23	**0,95**	**0,95**	**0,90**	-0,29	-0,26	0,31	0,26	0,37	-0,30	-0,28
90-PDE	**0,76**	**1,00**	0,49	-0,24	-0,29	**0,88**	**1,00**	**0,85**	-0,25	-0,27	0,31	0,27	0,38	-0,25	-0,23

Desses dados, notamos que as características dos grafos que impactam na execução dos algoritmos varia de acordo com: *(i)* o algoritmo (e.g, na CPU32, enquanto para o BFS as características mais relevantes são *Diâ.* e *90-PDE*, para o TC nesta mesma máquina são o *#Arestas* e *NT*); e *(ii)* a máquina (e.g, para o SSSP na CPU32 o *Diâ.* e *90-PDE* são mais relevantes, enquanto que na GPU2496 as mais relevantes são *#Arestas* e *NT*). Contudo, existe similaridade na distribuição de altos índices quando se compara as execuções nas CPU (CPU32 e CPU128), sendo o *Diâ.* e *90-PDE* sendo destacados para os algoritmos BFS e SSSP, e *#Arestas* e *NT* para algoritmos PR e TC. A exceção é para o algoritmo BC, onde na CPU32 as características mais relevantes foram *#Arestas* e *NT*, e na CPU128 foram o *#Ares. M. WCC*, *Diâ.*, e *90-PDE*.

No caso da GPU2496, nota-se que uma diferença nas características relevantes: os *#Arestas* e *NT* são as únicas características destacadas em todos os algoritmos. Por mais que essas tabelas por si só não apontem para um sentido de escolher a melhor máquina para executar uma aplicação de grafo específica, ela nos indica que, em geral, grafos com maior número de arestas/triângulo não sejam indicados para GPUs, o que pode ser explicado pela limitação da memória.

(ii) Performance nas Diferentes Máquinas. As Figuras 3 e 4 apresentam os tempos de execução (eixo y) dos algoritmos quando processando cada grafo (eixo x). Figura 3 e 4 apresentam os resultados normalizados pela CPU32 e CPU128, respectivamente (menor, melhor). Os resultados do PR são omitidos, uma vez que ele foi o único algoritmo que apresentou melhor performance em todos os grafos sempre quando executado em CPU (2544,66$\times$ e 1591,14$\times$ melhor na CPU32 e CPU128).

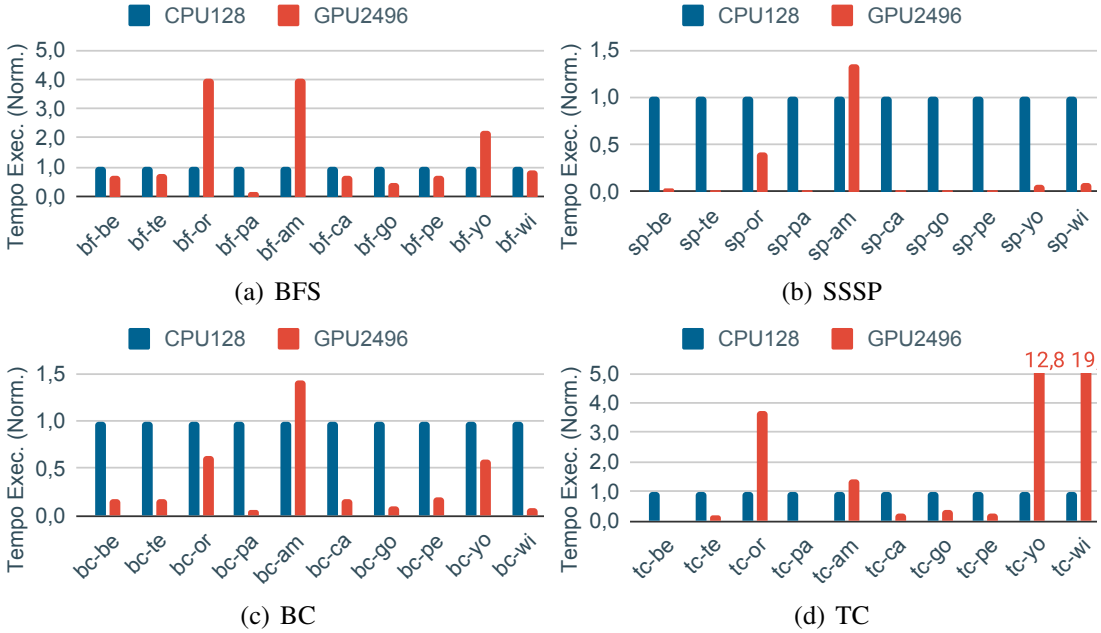
Com base nesses resultados, nota-se que em nenhum desses algoritmos é sempre melhor processar os grafos só em GPU ou em CPU. Por exemplo, apesar de para o SSSP, na maioria dos casos, ser melhor executado na GPU, processar o grafo orkut com SSSP na CPU32 implica em uma redução de 80% no tempo de execução (ver Figura 3b). No caso específico do orkut, é notório que na CPU32 ele é processado mais rapidamente por todos os algoritmos (e pelo BFS e TC na CPU128). Assim, dada essa diferença significativa entre as execuções em GPU e CPU, faz-se necessária uma análise mais aprofundada para possibilitar uma escolha apropriada de aplicação/máquina, o que mostraremos na próxima seção.

(iii) Extraindo as Características Relevantes. Nessa seção, juntamos os métodos de análise das seções anteriores, realizando a correlação dos *speedups* das CPUs em relação a GPU com as características de alto nível dos grafos. Na Tabela 4 temos alguns destaques em comum: alta correlação negativa do *Diâ.* e *90-PDE* com o algoritmo SSSP e correlação positiva no *#Vér. M. SCC* e *#Ares. M. SCC* com os algoritmos BFS, BC e TC. Com esses resultados, conseguimos extrair as características mais relevantes dos grafos que possam ser usadas em técnicas para tomada de decisão de onde processar os grafos,

(a) BFS (b) SSSP

(c) BC (d) TC

Figura 3. Comparação dos tempos de execução na CPU32 e na GPU2496. Os resultados são normalizados pela execução na CPU32 (menor, melhor).



(a) BFS (b) SSSP

(c) BC (d) TC

Figura 4. Comparação dos tempos de execução na CPU128 e na GPU2496. Os resultados são normalizados pela execução na CPU128 (menor, melhor).

seja em GPU ou em CPU.

(iv) Aplicação de Regressão Linear. Para isso, nós extraímos a característica mais relevante obtida na seção anterior para cada algoritmo que irá processar os grafos (e.g., *#Ares. M. SCC* para o BFS e *90-PDE* para o SSSP – ver na Tabela 4). Dos 10 grafos avaliados neste trabalho (ver Tabela 1), nós separamos 2 aleatoriamente (Youtube e Wikitalk) para teste, deixando os 8 restante para treino (20% para teste e 80% para treino). A Tabela 5 apresenta as decisões tomadas com base na Regressão Linear. Nela, nós destacamos em **negrito** os poucos casos onde a Regressão Linear falhou. No geral, usar apenas uma única característica do grafo para prever onde processá-lo resultou em uma acurácia média de 85%.

Tabela 4. Correlação entre os speedups das CPUs e as métricas dos grafos.

	GPU2496/CPU32					GPU2496/CPU128				
	BFS	**SSSP**	**BC**	**PR**	**TC**	**BFS**	**SSSP**	**BC**	**PR**	**TC**
#Vértices	-0,48	-0,22	**-0,67**	0,31	-0,23	-0,40	-0,09	**-0,55**	0,20	**-0,50**
#Arestas	0,25	0,39	0,12	0,47	0,13	0,38	0,33	0,22	**0,74**	0,04
#Vér. M. WCC	-0,16	-0,03	-0,18	0,12	**0,50**	0,24	0,02	0,14	0,32	0,17
#Ares. M. WCC	-0,25	0,27	-0,27	0,17	0,19	0,17	0,37	0,04	0,32	-0,01
#Vér. m. SCC	**0,67**	0,01	**0,81**	0,23	**0,53**	**0,69**	-0,20	**0,88**	0,45	**0,56**
#Ares. m. SCC	**0,78**	0,11	**0,88**	0,10	**0,62**	**0,74**	-0,13	**0,88**	0,38	**0,70**
CMC	0,12	**0,50**	0,23	-0,07	-0,30	0,13	**0,50**	0,15	-0,05	-0,05
NT	0,37	0,45	0,24	0,43	0,17	0,46	0,36	0,31	**0,74**	0,13
FTF	-0,11	0,01	0,13	-0,09	0,13	0,28	0,04	0,40	0,04	-0,01
Diâ.	0,38	**-0,69**	0,40	-0,24	-0,05	-0,17	**-0,84**	0,11	-0,37	0,09
90-PDE	0,32	**-0,77**	0,32	-0,27	0,18	-0,15	**-0,91**	0,09	-0,33	0,18

Tabela 5. Escolhas preditas pela regressão linear.

		BFS	**SSSP**	**BC**	**PR**	**TC**	**Acur.**
CPU32 ou	**Youtube**	**CPU**	GPU	GPU	CPU	CPU	80%
GPU2496	**Wikitalk**	GPU	GPU	GPU	CPU	**GPU**	80%
CPU128 ou	**Youtube**	CPU	GPU	GPU	CPU	CPU	100%
GPU2496	**Wikitalk**	GPU	GPU	GPU	CPU	**GPU**	80%
					Acur. Média		85%

6. Trabalhos Relacionados

Vários trabalhos na literatura já propuseram otimizar o processamento de algoritmos de grafos tanto em CPU quanto em GPU. Nessa seção, nós descrevemos os mais relevantes, separados em ambos os sistemas.

Processamento de Grafos em CPU. O trabalho de [Shun and Blelloch 2013] propõe Ligra, um modelo de programação de memória compartilhada para algoritmos de grafos que ajusta automaticamente a maneira como o grafo é avaliado de acordo com um limiar provido pelo programador. O trabalho de [Zhang et al. 2015] realiza um extenso estudo sobre os impactos da execução de algoritmos de grafos em sistemas NUMA. Com base nisso, eles propõem Polymer, um *framework* que otimiza o processamento baseando-se na localidade dos dados e em sua distribuição nas memórias do sistema. Amitabha Roy [Roy et al. 2013] propõe X-Stream, um *framework* para processamento de grafos que utiliza o modelo *scatter-gather* centrado em arestas. Por ser capaz de evitar acessos aleatórios à memória principal, X-Stream apresenta boa escalabilidade conforme o aumento no número de núcleos do sistema.

Processamento de Grafos em GPU. O trabalho de [Harish and Narayanan 2007] propõem um *framework* para processamento de grafos centrado no processamento de vértices, no qual os vértices são mapeados para as *threads* da GPU. O trabalho de [Hong et al. 2011] propõe o *Maximum warp*, uma estratégia que decompõe as *warps* da GPU em *sub-warps* com o objetivo de minimizar a divergência de caminho e melhorar o balanceamento de carga. Já a estratégia proposta por [Khorasani et al. 2014], o CuSha, resolve questões de eficiência através de duas representações de grafos, denominadas de G-shard e *concatenated windows*. Gunrock [Wang et al. 2016] é uma abstração para programação de algoritmos de grafos que usa o conceito de fronteiras, similar a outros *frameworks* de processamento de grafos em CPU [Shun and Blelloch 2013, Zhang et al. 2015].

Vários outros trabalhos implementam otimização de algoritmos específicos, os quais incluem o BFS baseado em filas hierárquicas ou soma prefixada [Luo et al. 2010],

CC [Greiner 1994], SSSP [Davidson et al. 2014] e BC [Jia et al. 2012].

Nossas Contribuições. Por mais que diferentes trabalhos já tenham otimizado o processamento de aplicações de grafos tanto em GPU quanto em CPU, nenhum deles se baseia nas características de alto nível dos grafos para decidir onde é mais vantajoso processá-los, se na GPU ou CPU. Além disso, nosso trabalho ortogonaliza os *frameworks* de GPU e CPU existentes, uma vez que nossa metodologia se baseia em tomar decisão com base nas características dos grafos e seu processamento em diferentes sistemas, independente de qual *framework* os algoritmos foram implementados.

7. Conclusão

Nesse trabalho, nós propomos uma metodologia que usa essas características de alto nível dos grafos (e.g., diâmetro e coeficiente de clusterização) para auxiliar na tomada de decisão da execução dos algoritmos de grafos em GPU ou CPU. Nossos resultados experimentais, considerando 1 GPU e 2 CPUs, mostraram que as características relevantes variam conforme os algoritmo que processarão os grafos e as máquinas onde serão executados. Além disso, como estudo de caso, nossa proposta apresentou uma acurácia média de 85% quando aplicando uma Regressão Linear Simples com as características dos grafos mais relevantes. Por fim, para trabalhos futuros, pretendemos expandir o conjunto de máquinas e analisar o impacto da nossa proposta com relação a outros requisitos não funcionais (e.g., energia e dissipação de potência).

Agradecimentos

Esse estudo foi financiado em parte pela CAPES - Finance Code 001, FAPERGS e o CNPq. Alguns experimentos desse trabalho usaram a infraestrutura do PCAD, http://gppd-hpc.inf.ufrgs.br/, do INF/UFRGS.

Referências

Beamer, S., Asanović, K., and Patterson, D. (2015). The gap benchmark suite. *arXiv preprint arXiv:1508.03619*.

Benesty, J., Chen, J., Huang, Y., and Cohen, I. (2009). Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer.

Davidson, A., Baxter, S., Garland, M., and Owens, J. D. (2014). Work-efficient parallel gpu methods for single-source shortest paths. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 349–359. IEEE.

Davis, T. A. (2019). Algorithm 1000: Suitesparse: Graphblas: Graph algorithms in the language of sparse linear algebra. *ACM Transactions on Mathematical Software (TOMS)*, 45(4):1–25.

de A. Rocha, H. M. G., Schwarzrock, J., Lorenzon, A. F., and Beck, A. C. S. (2022). Using machine learning to optimize graph execution on numa machines. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 1027–1032.

Greiner, J. (1994). A comparison of parallel algorithms for connected components. In *Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures*, pages 16–25.

Harish, P. and Narayanan, P. J. (2007). Accelerating large graph algorithms on the gpu using cuda. In *International conference on high-performance computing*, pages 197–208. Springer.

Hong, S., Kim, S. K., Oguntebi, T., and Olukotun, K. (2011). Accelerating cuda graph algorithms at maximum warp. *Acm Sigplan Notices*, 46(8):267–276.

Jia, Y., Lu, V., Hoberock, J., Garland, M., and Hart, J. C. (2012). Edge v. node parallelism for graph centrality metrics. In *GPU Computing Gems Jade Edition*, pages 15–28. Elsevier.

Khorasani, F., Vora, K., Gupta, R., and Bhuyan, L. N. (2014). Cusha: vertex-centric graph processing on gpus. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pages 239–252.

Leskovec, J. and Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`.

Lorenzon, A. F. and Beck Filho, A. C. S. (2019). *Parallel computing hits the power wall: principles, challenges, and a survey of solutions*. Springer Nature.

Lorenzon, A. F., Souza, J. D., and Beck, A. C. S. (2017). Laant: A library to automatically optimize edp for openmp applications. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1229–1232. IEEE.

Luo, L., Wong, M., and Hwu, W.-m. (2010). An effective gpu implementation of breadth-first search. In *Design Automation Conference*, pages 52–55. IEEE.

Mofrad, M. H., Melhem, R., Ahmad, Y., and Hammoud, M. (2020). Graphite: a numa-aware hpc system for graph analytics based on a new mpi* x parallelism model. *Proceedings of the VLDB Endowment*, 13(6):783–797.

Moori, M. K., Rocha, H. M. G. d. A., Schwarzrock, J., Lorenzon, A. F., and Beck, A. C. S. (2021). Aumentando a eficiência na execução de algoritmos de grafos em hpc. In *Anais do XXII Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 132–143. SBC.

Rocha, H. M. G. d. A., Schwarzrock, J., Lorenzon, A. F., and Beck, A. C. S. (2021). Boosting graph analytics by tuning threads and data affinity on numa systems. In *PDP*, pages 161–168. IEEE.

Roy, A., Mihailovic, I., and Zwaenepoel, W. (2013). X-stream: Edge-centric graph processing using streaming partitions. page 472–488.

Sahu, S., Mhedhbi, A., Salihoglu, S., Lin, J., and Özsu, M. T. (2020). The ubiquity of large graphs and surprising challenges of graph processing: extended survey. *The VLDB Journal*, 29(2):595–618.

Shun, J. and Blelloch, G. E. (2013). Ligra: a lightweight graph processing framework for shared memory. In *ACM PPoPP*, pages 135–146.

Staudt, C. L., Sazonovs, A., and Meyerhenke, H. (2016). Networkit: A tool suite for large-scale complex network analysis. *Network Science*, 4(4):508–530.

Sun, J., Vandierendonck, H., and Nikolopoulos, D. S. (2017). Graphgrind: Addressing load imbalance of graph partitioning. In *Proceedings of the International Conference on Supercomputing*, pages 1–10.

Wang, Y., Davidson, A., Pan, Y., Wu, Y., Riffel, A., and Owens, J. D. (2016). Gunrock: A high-performance graph processing library on the gpu. In *Proceedings of the 21st ACM SIGPLAN symposium on principles and practice of parallel programming*, pages 1–12.

Yang, C., Buluç, A., and Owens, J. D. (2022). Graphblast: A high-performance linear algebra-based graph framework on the gpu. *ACM Transactions on Mathematical Software (TOMS)*, 48(1):1–51.

Zhang, K., Chen, R., and Chen, H. (2015). Numa-aware graph-structured analytics. *SIGPLAN Not.*, 50(8):183–193.