AVALIAÇÃO DE DESEMPENHO DO ALGORITMO PAGERANK SEQUENCIAL E PARALELO

João Vitor Spolavore Thiago Gonçalves

1 de dezembro de 2025

O presente relatório tem como objetivo analisar o comportamento do sistema ao aplicar o algoritmo PageRank sob diferentes níveis de paralelismo, avaliando como o aumento do número de threads impacta seu desempenho.

Todos os experimentos realizados foram executados no Parque de Computação de Alto Desempenho (PCAD) da Universidade Federal do Rio Grande do Sul (UFRGS). Utilizamos o nó computacional denominado *blaise*.

- **Nome:** blaise
- **CPU:** 2 x Intel(R) Xeon(R) E5-2699 v4, 2.20 GHz, 88 threads, 44 cores
- **RAM:** 256GB
- **Disco:** 1.8 TB SSD, 1.8 TB HDD
- **Placa mãe:** Supermicro X10DGQ

`https://gppd-hpc.inf.ufrgs.br`

Esta máquina possui DOIS processadores Intel, fundamental para a escolha devido às possibilidades de experimentos com distribuição de threads.

O presente projeto apresenta uma perspectiva de medição para o método de análise.

Para ajudar nessa abordagem, utilizamos o Intel Vtune Profiler para coletar métricas de hardware da aplicação. Decidimos utilizar essa ferramenta pois:

- Os processadores presentes na máquina escolhida eram Intel.
- É uma ferramenta amplamente utilizada pela própria Intel para obter métricas de seus processadores.

Para obter o tempo de execução da programa (*Elapsed time*) não utilizamos o profiler da Intel. Uma vez que a sua utilização adiciona um *overhead* significativo nessa métrica.

Executamos cada configuração 5 vezes utilizando o Vtune e 10 vezes sem a utilização do profiler.

Consideramos as seguintes opções para análise:

- **Número de Threads**: 1, 4, 12, 22, 28, 36, 44, 66, 88.
- **Estado do Hyperthreading (ON/OFF)**: Com o objetivo de avaliar o impacto no desempenho, forçamos 1 thread por core com a variável `OMP_PLACES=cores/threads` e reduzimos visibilidade de threads lógicas com `GOMP_CPU_AFFINITY` e o comando `taskset -c [NÚMEROS DOS CORES]`.
- **Política de *Binding* de Threads**:Distribuição de threads entre os núcleos dos processadores via `OMP_PROC_BIND` com essas opções:
  - `close`: Vincula threads a lugares próximos à thread pai, útil para localidade de cache.
  - `spread`: Distribui threads em partições mais distantes, útil para reduzir contenção em recursos compartilhados.

- **Algoritmo:** Utilizamos uma implementação consolidada do PageRank do repositório GAPBS (`https://github.com/sbeamer/gapbs`).
- **Fonte dos Dados:** Os grafos foram obtidos da *Stanford Large Network Dataset Collection* (SNAP), um repositório consagrado academicamente (`https://snap.stanford.edu/data/index.html`).
- **Requisito:** Para executar o algoritmo PageRank, a aplicação recebe como entrada grafos **direcionados**.

A tabela abaixo detalha as redes de grafos escolhidas para os experimentos:

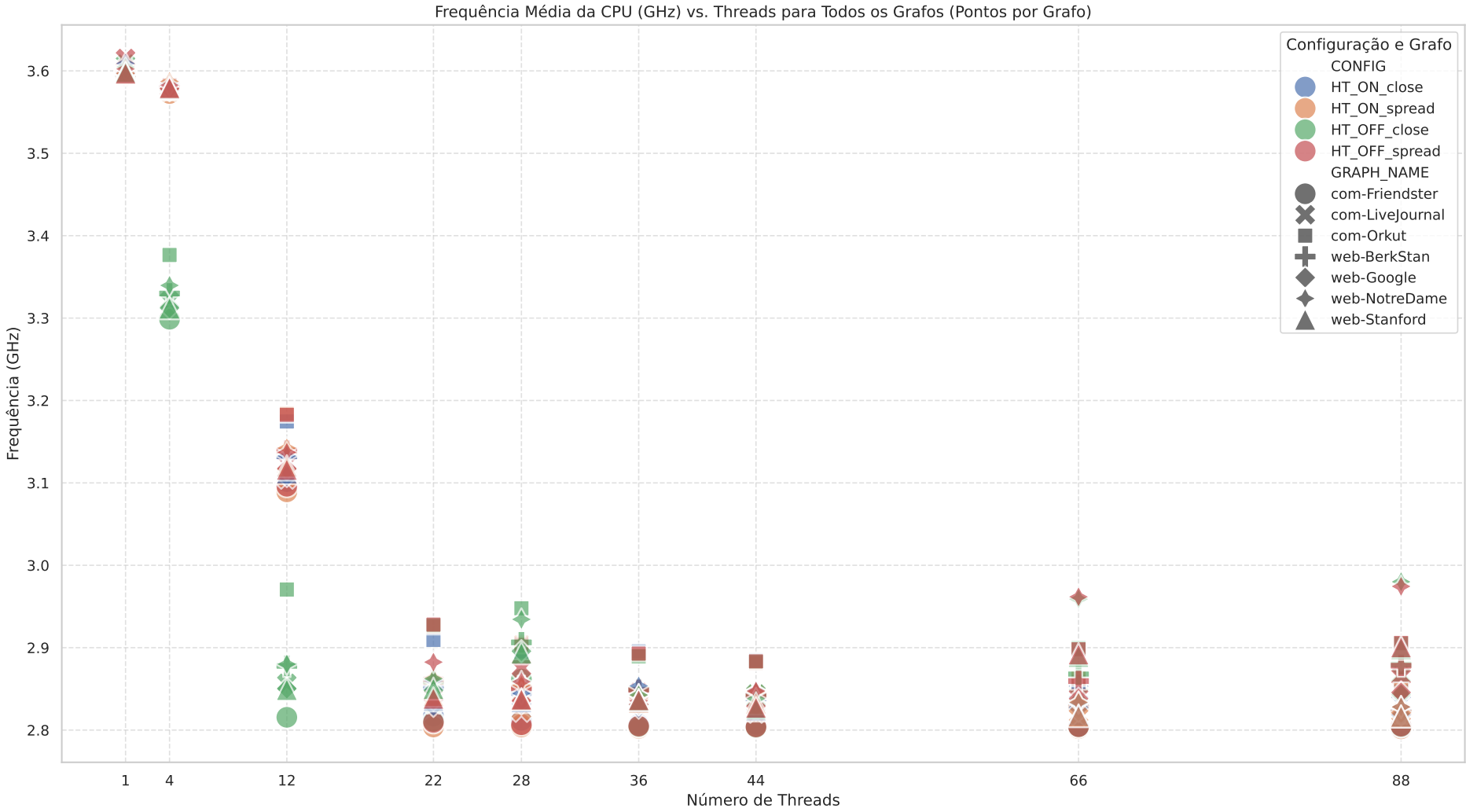
Grafo	**Vértices**	**Arestas**
Friendster	65,608,366	1,806,067,135
LiveJournal	3,997,962	34,681,189
Orkut	3,072,441	117,185,083
BerkStan	685,230	7,600,595
Google	875,713	5,105,039
NotreDame	325,729	1,497,134
Stanford	281,903	2,312,497

Todos representam conexões entre páginas web e/ou redes sociais e são dados do mundo real.

Primeiro, mostraremos as métricas brutas do *Intel Vtune Profiler*.
Vamos iniciar vendo o gráfico de Frequência Média da CPU durante a execução do algoritmo.
Nesse gráfico, os pontos representam todos os grafos de entrada, com as 4 seguintes configurações:

1. Hyperthreading Desligado, Política Spread - **HT_OFF_spread**
2. Hyperthreading Ligado, Política Spread - **HT_ON_spread**
3. Hyperthreading Desligado, Política Close - **HT_OFF_close**
4. Hyperthreading Ligado, Política Close - **HT_ON_close**

FREQUÊNCIA MÉDIA DA CPU



Como o tempo de nossa apresentação é limitado, apenas apresentaremos a análise para uma entrada da aplicação.
Agora mostraremos as tabelas de *Memory Bound* e ciclos por instrução.
Memory Bound é a porcentagem de tempo perdido esperando operações na memória.
Ciclos por instrução (CPI) é uma média total de ciclos pra cada instrução do programa.

TABELA DE *MEMORY BOUND* (%)

GRAPH_NAME	CONFIG	THREADS								
		1	4	12	22	28	36	44	66	88
Friendster	H_OFF_C	29.20	33.08	41.74	56.46	66.58	70.16	73.02	73.30	76.84
	H_OFF_S	29.26	40.62	52.08	62.50	67.56	70.90	73.06	73.66	76.20
	H_ON_C	33.50	21.56	19.72	23.86	31.62	37.00	39.58	52.86	68.36
	H_ON_S	28.34	37.18	53.38	61.76	64.26	69.26	71.50	61.56	68.00

- Nenhuma única entrada obteve a maior ou menor taxa de ciclos ociosos esperando por operações na memória, sendo um padrão que se repete ao longo de todas as entradas.
- Além disso, por mais que seja possível observar que nesse caso os melhores *Memory Bound* estavam com menos *threads* e os piores com mais, esse comportamento não se repete com todas as entradas.

TABELA DE CICLOS POR INSTRUÇÃO

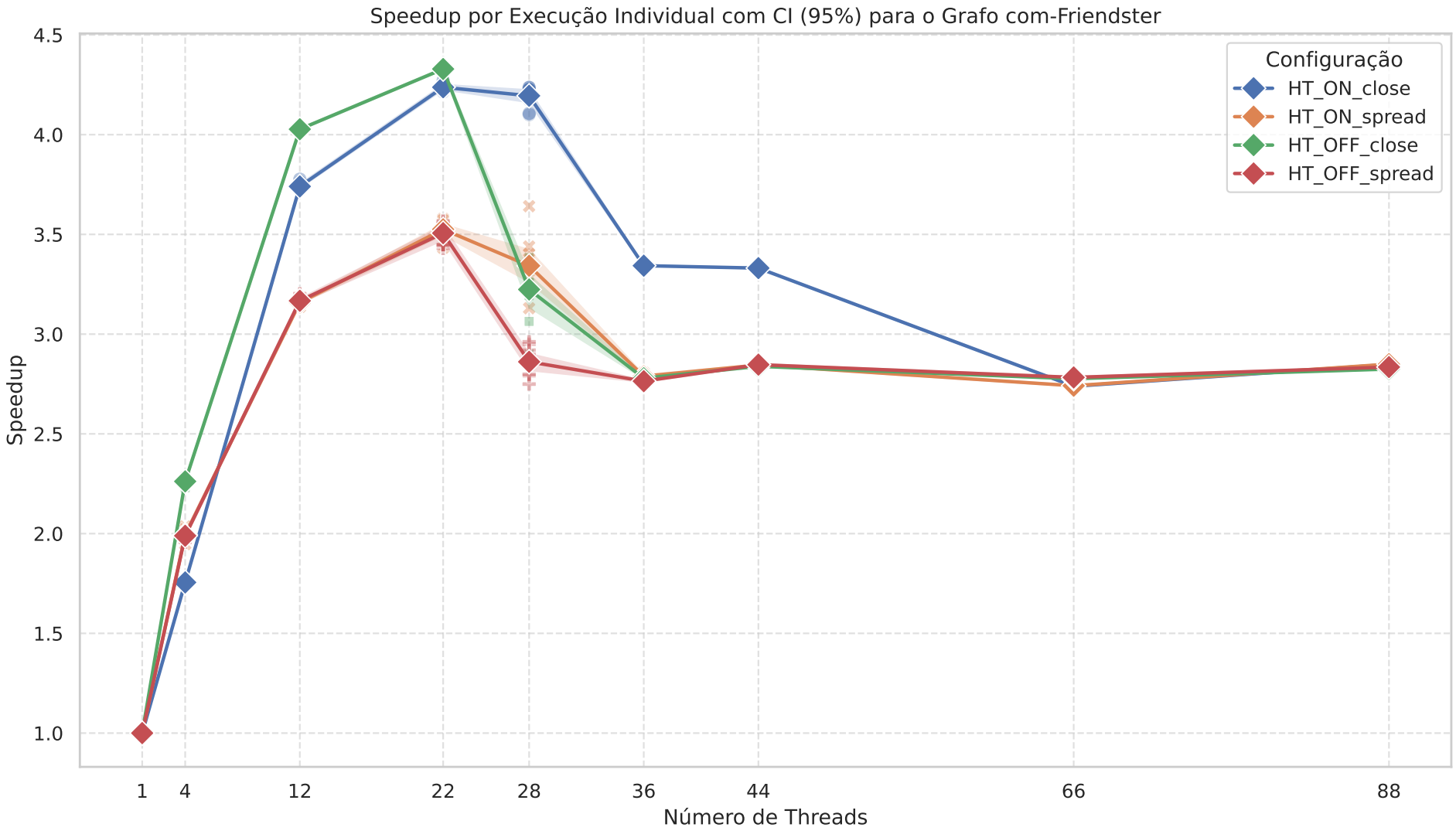
GRAPH_NAME	CONFIG	THREADS								
		1	4	12	22	28	36	44	66	88
Friendster	H_OFF_C	0.663	0.697	0.861	1.314	2.020	2.590	3.104	2.975	3.233
	H_OFF_S	0.667	0.850	1.173	1.692	2.058	2.560	3.113	2.962	3.226
	H_ON_C	0.792	1.148	1.202	1.330	1.592	1.925	2.302	3.830	5.187
	H_ON_S	0.612	0.730	1.101	1.539	1.843	2.375	2.995	4.993	5.845

- O comportamento de ciclos por instrução se repetiu ao longo de todas as entradas.
- O grafo Friendster foi uma exceção, com os outros grafos possuindo o maior CPI com o *Hyperthreading* desligado ao executar com 44 *threads*.
- Achamos que pode ser por causa do grande tamanho do grafo e altos volumes de transferência de dados.

O gráfico a seguir mostra o *speedup* para o grafo com-Friendster, o maior grafo em que realizamos nossos experimentos, com intervalo de confiança de 95% entre cada ponto.
Nesse gráfico, os pontos representam todos os grafos de entrada, com as 4 seguintes configurações:

1. **HT_OFF_spread**
2. **HT_ON_spread**
3. **HT_OFF_close**
4. **HT_ON_close**

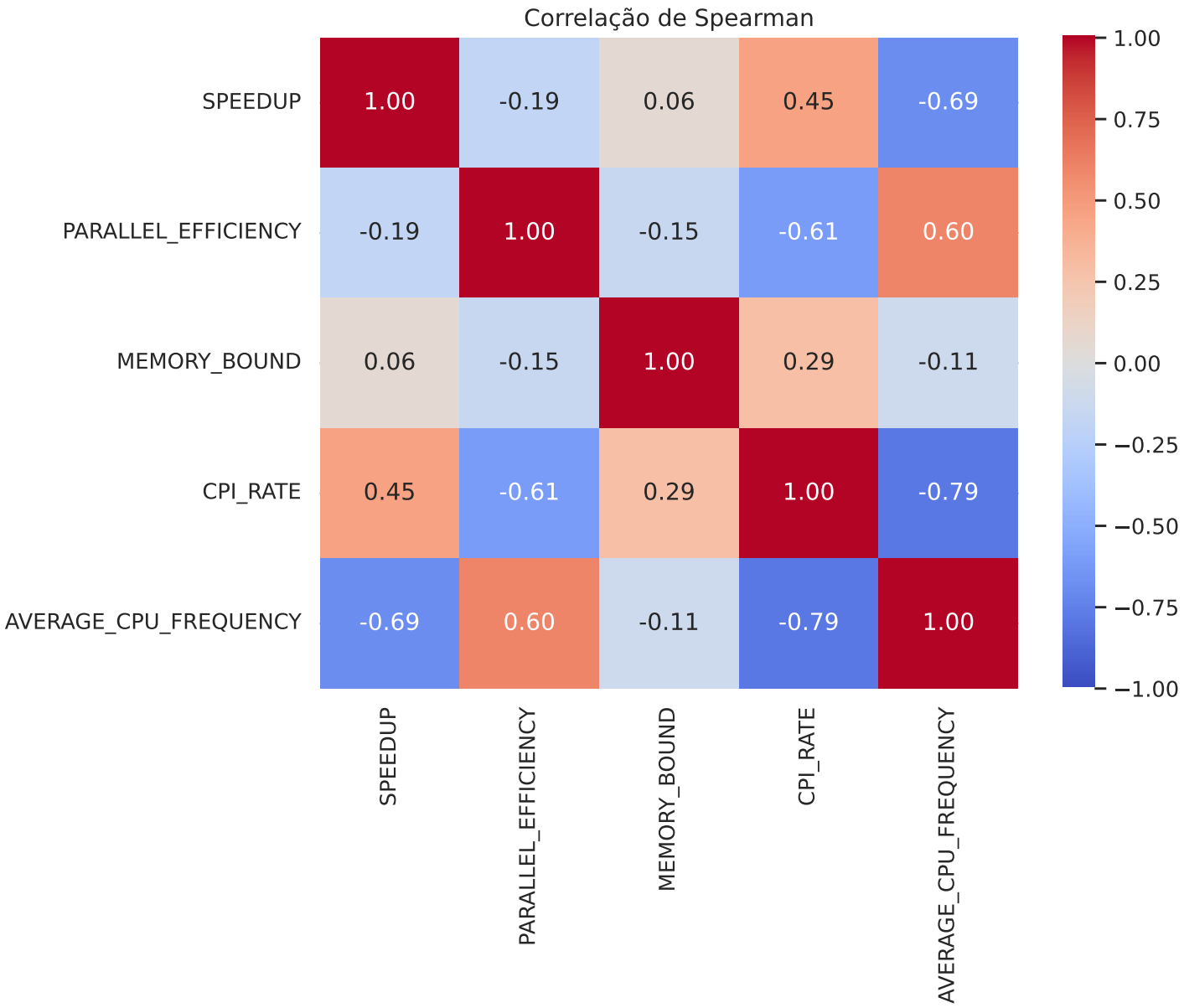
SPEEDUP - FRIENDSTER



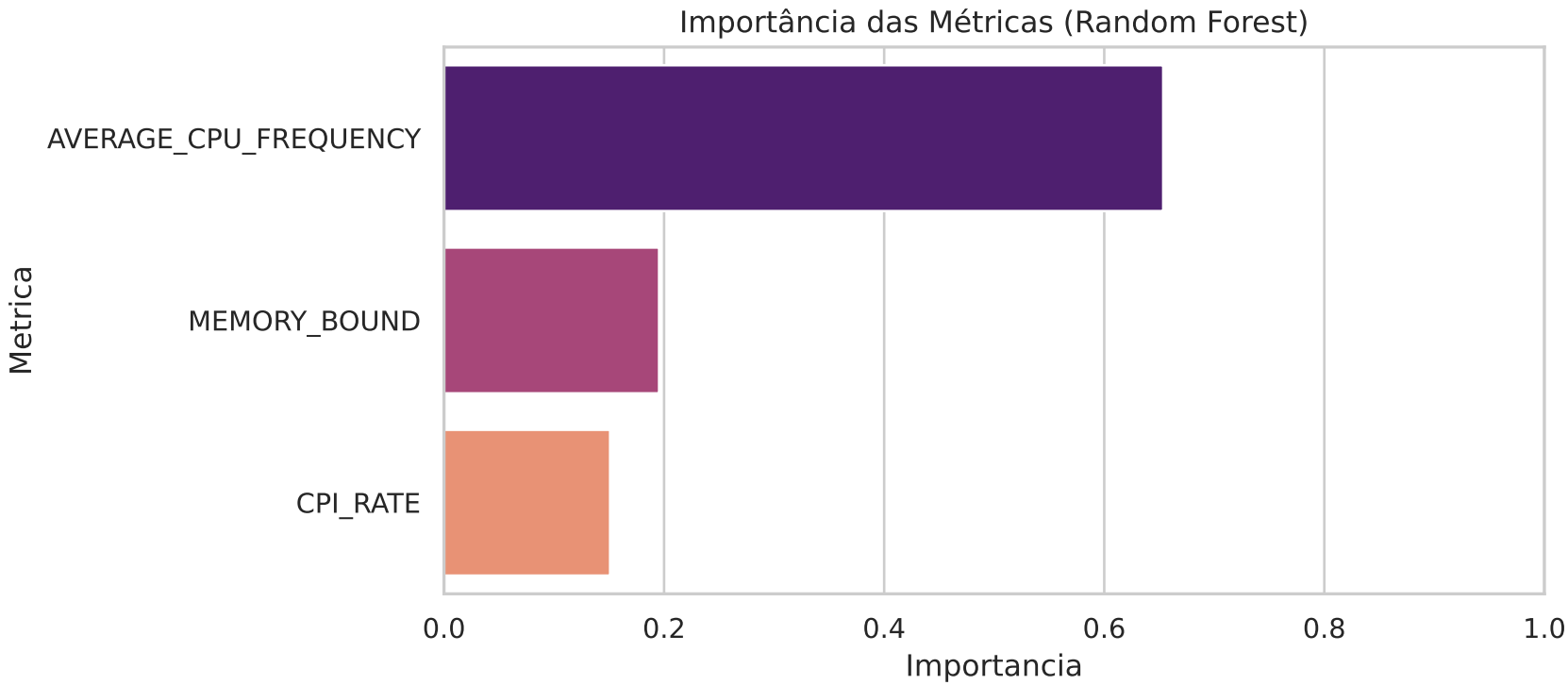
SPEEDUP - FRIENDSTER

- Melhor caso executando com *Hyperthreading* desligado e política *close* - visto também em outras entradas.
- Diferente de outras entradas, o caso com *Hyperthreading* liagdo e política *close* tambem se saiu bem. Achamos que pode ser relacionado com a eficiência térmica em grafos maiores.

ANÁLISE ESTATÍSTICA - CORRELAÇÃO SPEARMAN



ANÁLISE ESTATÍSTICA - *FEATURE IMPORTANCE*



- A frequência da CPU é o que mais importa pro *speedup* (estando correlacionada negativamente).
- Atrasos na memória afetam mais o *speedup* do que o CPI.

Por fim, fizemos uma análise com ANOVA, tentando associar nossas variáveis de entrada com as saídas do *Intel Vtune*
Assim, resultados com *p-value* menor que 0.05 (5%) estão marcados em verde e significam que a variável possui um impacto estatisticamente signficativo naquela métrica.

	sum_sq	F	p-value
C(THREADS)	21.194140	764.778120	0.000000
C(HT_OFF)	0.009423	2.720265	0.100390
C(BINDING)	0.040890	11.804060	0.000696
C(HT_OFF):C(BINDING)	0.041835	12.076720	0.000606
Residual	0.831384	NaN	-

ANOVA pra Frequência média da CPU

	sum_sq	F	p-value
C(THREADS)	48.361651	18.586491	0.000000
C(HT_OFF)	7.106524	21.849600	0.000005
C(BINDING)	0.033739	0.103734	0.747674
C(HT_OFF):C(BINDING)	0.294290	0.904820	0.342450
Residual	78.059354	NaN	-

ANOVA pra taxa de ciclos por instrução

	sum_sq	F	p-value
C(THREADS)	5300.901527	2.754524	0.006290
C(HT_OFF)	3510.528229	14.593494	0.000170
C(BINDING)	2063.090006	8.576399	0.003733
C(HT_OFF):C(BINDING)	672.672057	2.796342	0.095783
Residual	57733.038537	NaN	-

ANOVA pra taxa de *Memory Bound*

	sum_sq	F	p-value
C(THREADS)	191.385561	17.909454	0.000000
C(HT_OFF)	0.508229	0.380472	0.537936
C(BINDING)	1.135873	0.850341	0.357382
C(HT_OFF):C(BINDING)	0.734943	0.550195	0.458963
Residual	320.588597	NaN	-

ANOVA pro *Speedup*

Este trabalho apresentou uma análise detalhada do desempenho do algoritmo PageRank em arquiteturas modernas, explorando o impacto de configurações de multithreading, afinidade de processador e microarquitetura no tempo de execução.

Nossa análise demonstrou que para algoritmos limitados por memória como o PageRank a estratégia padrão de usar o máximo de threads lógicas disponíveis nem sempre resulta no melhor desempenho.

Tivemos algumas ideias para melhorar o desempenho da nossa aplicação, como:

1. Reduzir o tempo gasto por atrasos na memória. Isso poderia ser feito utilizando um processador com barramento maior, ou com memória *cache* maior.
2. Utilizar uma máquina mais atual que a blaise. O processador dela foi lançado a 9 anos atrás, e tecnologias mais recentes (ex. *3D VCache* da AMD) podem melhorar o desempenho dessa aplicação.

DIFICULDADES

- Em diversos momentos a máquina utilizada (blaise) estava alocada por outra pessoa.
- Organizar os dados gerados pelo Intel Vtune Profiler. Ele gera muitos dados, e a maioria não era de interesse do nosso grupo.
- Google Colab não apresentou um *pair programming* satisfatório.

- A importâncida da aplicação de um projeto experimental sólido e replicável.
- Utilização de análise estatística criteriosa para verificar correlações entre os parâmetros do experimento.

OBRIGADO!

João Vitor Spolavore **Thiago Gonçalves**
Instituto de Informática — UFRGS

Todos dados disponíveis no repositório: `https://github.com/thgdsg/perf-analysis`