COMPARAÇÃO ENTRE PAGERANK SEQUENCIAL E PARALELIZADO

João Vitor Spolavore Thiago Gonçalves

6 de outubro de 2025

AGENDA

1. Introdução
2. Ambiente de Teste
3. Método de análise e coleta de dados
4. Algoritmo e Dados
5. Resultados
6. Dificuldades
7. Conclusão
8. Próximos passos

INTRODUÇÃO

O presente relatório tem por objetivo apresentar o efetivo progresso realizado pelos estudantes João Vitor Spolavore e Thiago Gonçalves da análise comparativa entre o algoritmo PageRank Sequencial e sua versão Paralela.

Ainda, explicaremos como o projeto foi realizado, as decisões tomadas, as escolhas de parâmetros, os resultados até o momento e, por fim, citar as dificuldades enfrentadas ao longo do projeto.

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.

Como citado na Etapa 1, o presente projeto apresenta uma perspectiva de medição para o método de análise.

Para ajudar nessa abordagem, utilizamos o Intel Vtune Profiler para coletar os dados da aplicação - gerando um relatório em formato *csv* ao final da execução. Decidimos utilizar essa ferramenta pois:

- O grupo já possuia uma breve experiência com ela.
- Os processadores presentes na máquina escolhida eram Intel.
- É uma ferramenta amplamente utilizada pela própria Intel para obter métricas de seus processadores.

ANÁLISE DO VTUNE - PONTOS POSITIVOS

- Fácil configuração.
  - 1 comando para carregar as variáveis.
  - 1 comando para realizar a coleta.
  - 1 comando para realizar a sumário em csv.
- Integração simples com o objeto de análise.
- Diretórios de output configuráveis.

ANÁLISE DO VTUNE - PONTOS NEGATIVOS

- Arquivos de saída mal formatados.
- Overhead considerável e diferente entre os tipos de análise.
- Excesso de arquivos gerados no final da análise.

Com isso tudo, consideramos que foi uma escolha adequada e facilitou consideravelmente o processo de coleta de dados.

Escolhemos utilizar o governor DVFS *performance* com o comando `cpufreq-set -g performance`.

A fim de centralizar todos as instâncias dos experimentos desejados e automatizar o processo de execução através de scripts, criamos um csv contendo a definição de todos os nossos experimentos denominado experiments.csv.

Além disso, executamos cada configuração 5 vezes pra garantir a robustez dos nossos resultados e também evitar o impacto de possíveis outliers na nossa pesquisa. Por fim, para essa etapa focamos nas métricas de tempo total gasto, speed up e eficiência do paralelismo.

- **Quantidade de instâncias:** 757.
- **Quantidade de execuções:** 757 * 5 = 3785.
- **Quantidade de colunas no arquivo:** 9.

Conteúdo completo do arquivo de experimentos:
`https://github.com/thgdsg/perf-analysis/blob/main/stage2/experiments.csv`

Consideramos as seguintes opções para análise:

- **Número de Threads**: 1, 4, 12, 22, 28, 36, 44, 66, 88. Abrange threads físicas (22), lógicas/totais por CPU (44), e total da máquina (88).
- **Tipo de Análise feita pelo Profiler**: hpc-performance, hotspots, performance-snapshot. Importante para obter informações sobre gasto de tempo, eficácia da paralelização e uso/atribuição de threads.

- **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**: Analisa o impacto da distribuição de threads entre os dois 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.

- **hpc-performance**: Essa análise coleta métricas detalhadas de uso de hardware, como largura de banda de memória, eficiência de cache, paralelismo de instruções (ILP) e balanceamento entre threads, ajudando a identificar gargalos em aplicações fortemente paralelas.
- **hotspots**: Essa análise é útil para detectar funções ou loops críticos e orientar otimizações de desempenho no nível do código-fonte.
- **performance-snapshot**: Fornece uma visão geral automática do desempenho da aplicação. Ele realiza uma coleta rápida de métricas-chave — como utilização da CPU, eficiência de paralelismo, uso de memória e E/S.

Scripts utilizados para automatizar execução e análise:

- `perf_analysis_pr.sh`: Orquestração local (prepara ambiente, carrega VTune, executa comandos).
- `run.slurm`: Execução em cluster SLURM.
- `src/build_commad.py`: Geração da matriz de execuções de `experiments.csv` para `src/commands.sh`.
- `src/commands.sh`: Script gerado com todas as combinações de execuções.
- `src/unify_all_results.py`: Pós-processamento (extrai tempo, calcula speedup e eficiência, unifica em `unified_results.csv`).
- `src/execute.sh`: Executa uma instância específica. Recebe os parâmetros presentes no experiments.csv, configura as variáveis de ambiente do OpenMP, criar os diretórios de output e chama o comando do vtune para coleta e sumário de dados.

Fluxo Resumido

1. Preparação do ambiente
2. Geração de comandos a partir de `experiments.csv`
3. Execução sistemática
4. Coleta em `results/`
5. Unificação em `unified_results.csv`

Entrada: `experiments.csv`.
Saídas: Diretório `results/` e `unified_results.csv` juntando todos resultados em um csv.
Visamos obter o máximo de reprodutibilidade, minimizar o impacto de *outliers*, e ter uma limpeza de resultados.

- **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.

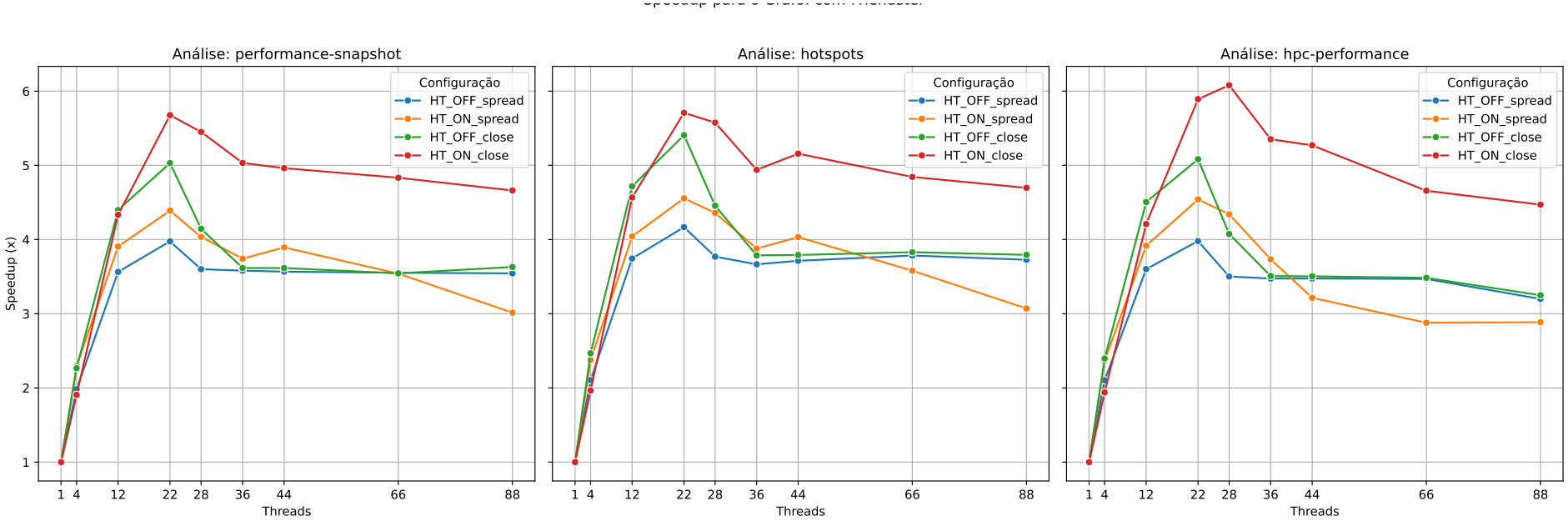
Agora, mostraremos os gráficos de *speedup*. O speedup é quão mais rápido a versão paralela foi comparado a versão sequencial.
Os gráficos a seguir estão organizados da esquerda pra direita:

- Tipo de Análise performance-snapshot
- Tipo de Análise hotspots
- Tipo de Análise hpc-performance

E com 4 possíveis organizações:

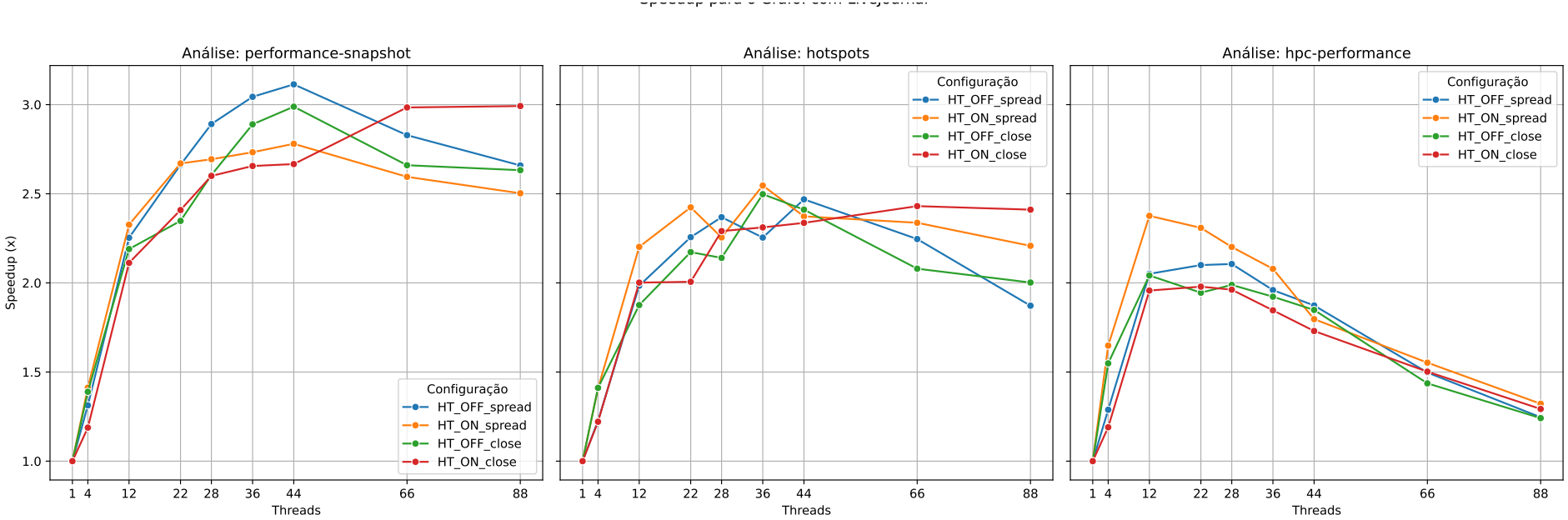
1. Hyperthreading Desligado, Política Spread - HT_OFF_spread
2. Hyperthreading Ligado, Política Spread - HT_ON_spread
3. Hyperthreading Desligado, Política Close - HT_OFF_close
4. Hyperthreading Ligado, Política Close - HT_ON_close

SPEEDUP - FRIENDSTER



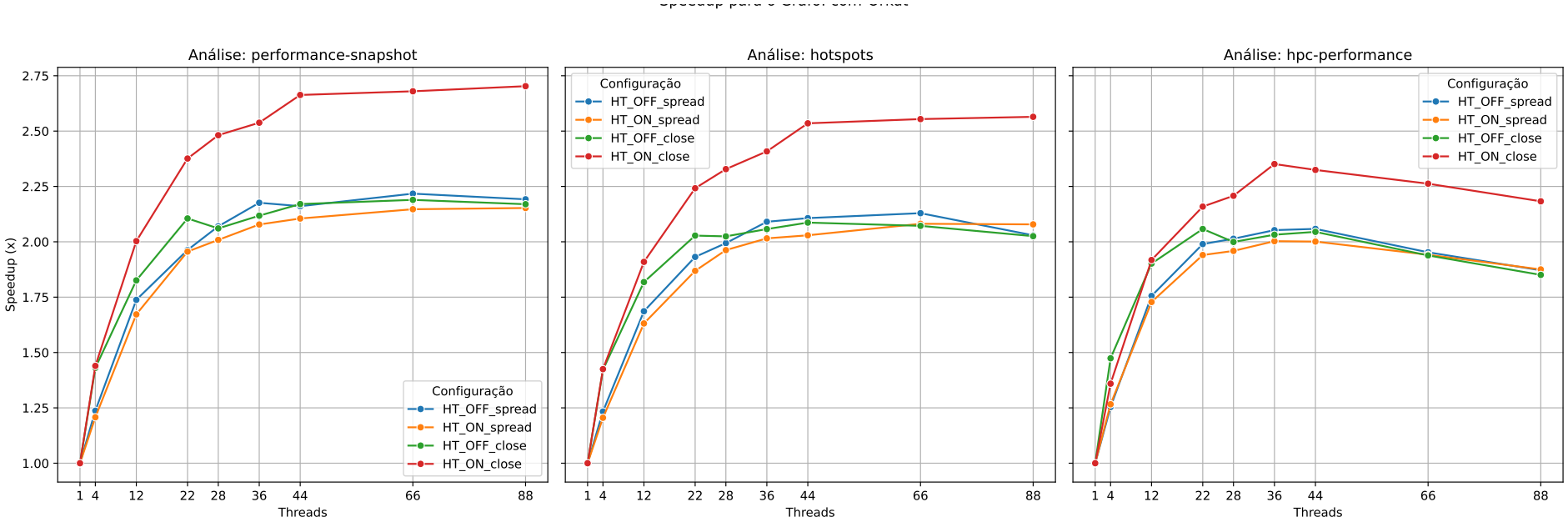
No maior grafo, o Hyperthreading Ligado com a política "close"(HT_ON_close) obteve melhor speedup em quase todos os casos.

SPEEDUP - LIVEJOURNAL



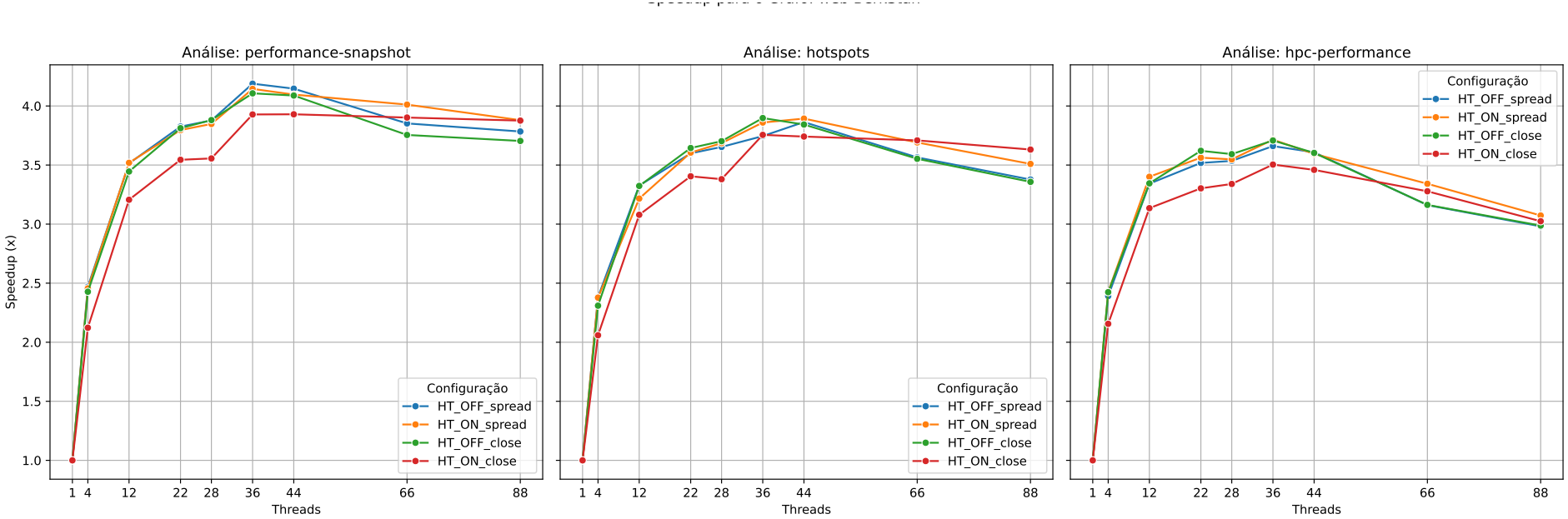
Nesse grafo de tamanho mais modesto, as outras configurações além da HT_ON_close conseguiram obter speedup maior com menos threads.

SPEEDUP - ORKUT



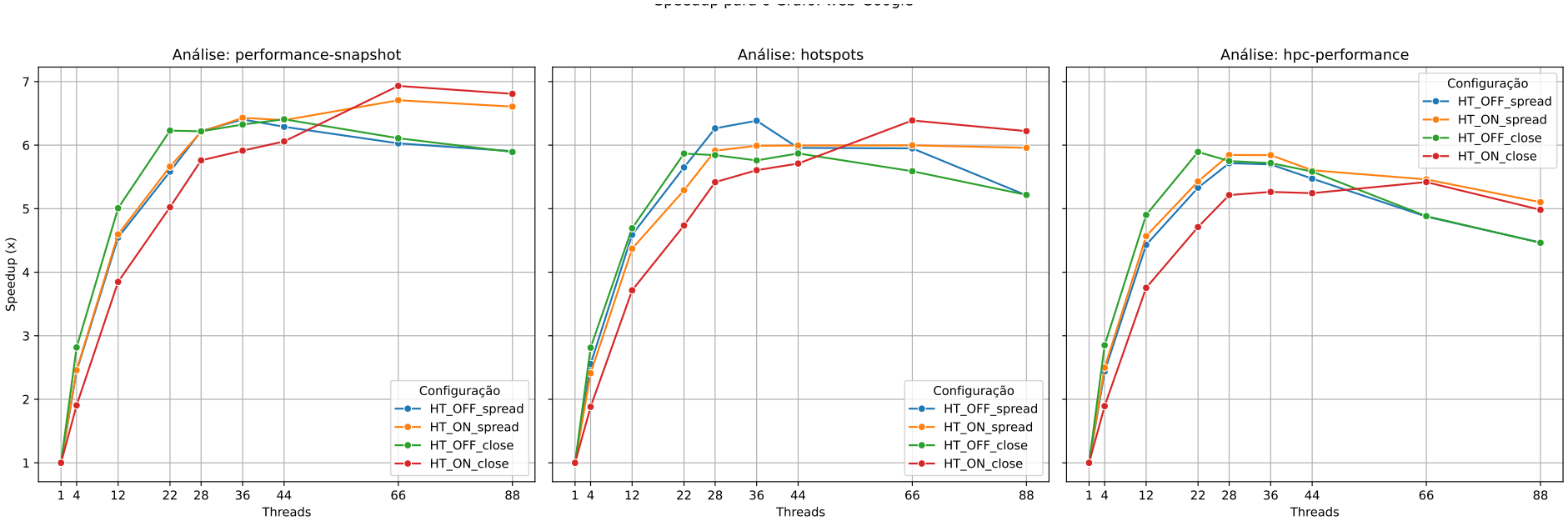
A configuração HT_ON_close foi melhor em todos os casos.

SPEEDUP - BERKSTAN



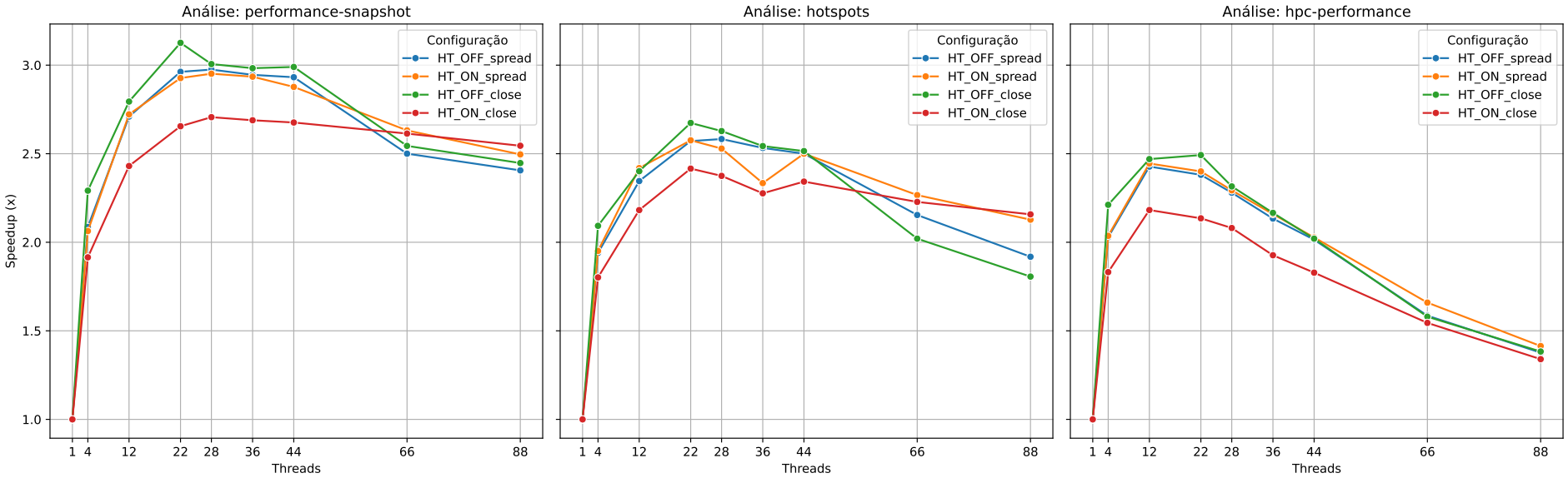
Resultados mistos, não houve nenhuma que se destacou, mas o speedup da HT_ON_close foi menor com menos threads.

SPEEDUP - GOOGLE



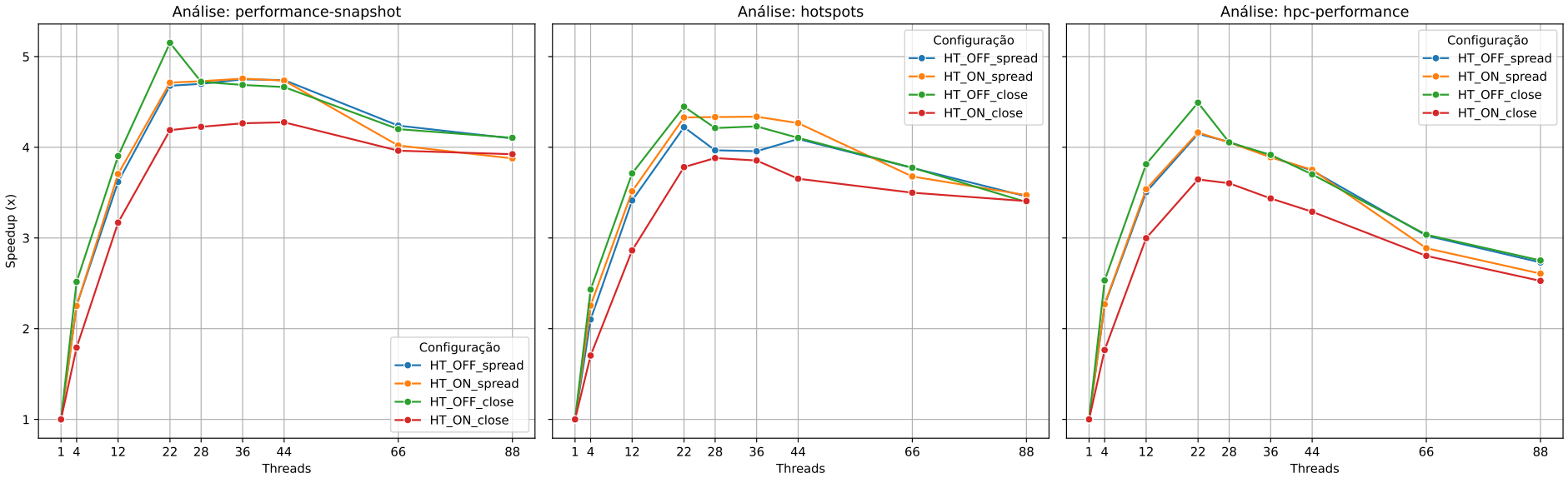
Novamente, a HT_ON_close se saiu melhor com mais threads.

SPEEDUP - NOTREDAME



A HT_OFF_close se saiu muito bem com poucas threads, obtendo o maior speedup.

SPEEDUP - STANFORD



Novamente, a HT_OFF_close obteve o melhor speedup com poucas threads.

MELHORES CONFIGURAÇÕES DE SPEEDUP (PARTE 1)

Grafo	Análise	Threads	HT	Bind	Speedup
com-Friendster	hotspots	22	True	close	5.71
	hpc	28	True	close	6.08
	snapshot	22	True	close	5.68
com-LiveJournal	hotspots	36	True	spread	2.55
	hpc	12	True	spread	2.38
	snapshot	44	False	spread	3.11
com-Orkut	hotspots	88	True	close	2.56
	hpc	36	True	close	2.35
	snapshot	88	True	close	2.70
web-BerkStan	hotspots	36	False	close	3.90
	hpc	36	True	spread	3.72
	snapshot	36	False	spread	4.19

MELHORES CONFIGURAÇÕES DE SPEEDUP (PARTE 2)

Grafo	Análise	Threads	HT	Bind	Speedup
web-Google	hotspots	66	True	close	6.39
	hpc	22	False	close	5.89
	snapshot	66	True	close	6.93
web-NotreDame	hotspots	22	False	close	2.67
	hpc	22	False	close	2.49
	snapshot	22	False	close	3.13
web-Stanford	hotspots	22	False	close	4.45
	hpc	22	False	close	4.49
	snapshot	22	False	close	5.15

Agora, mostraremos os gráficos de eficiência paralela. A eficiência paralela é uma maneira de quantificar o quão bem foram utilizados os recursos de *hardware* no paralelismo.
Ela pode ser calculada pela seguinte fórmula:

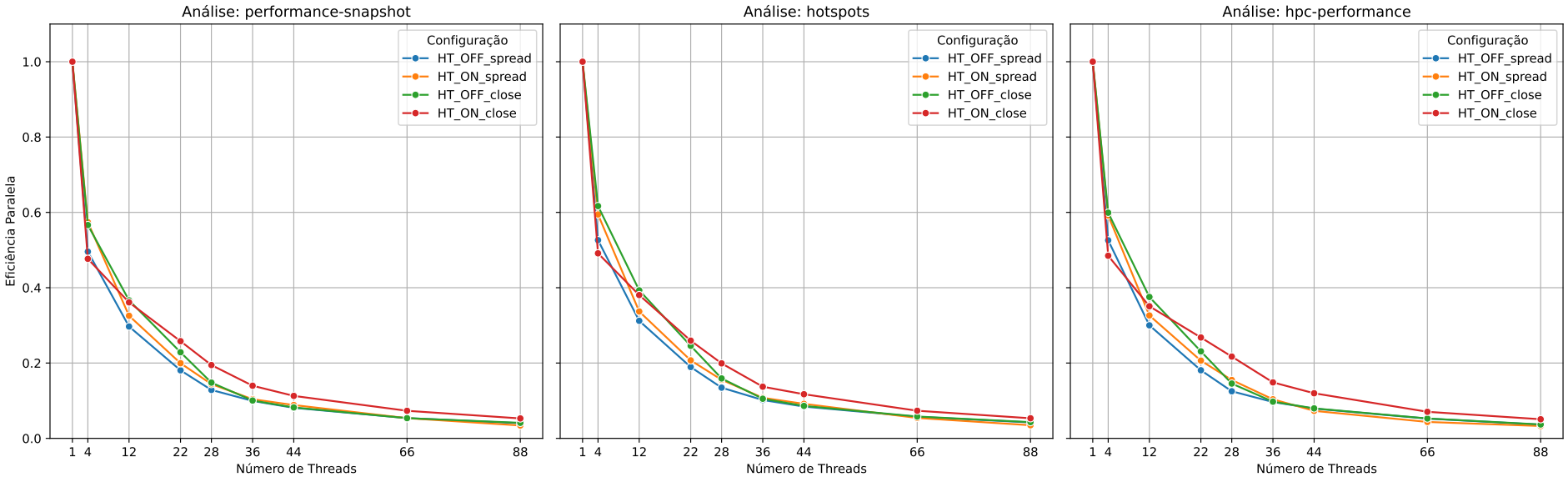
$$E_p = \frac{T_s}{(N \times T_p)}$$

Onde:

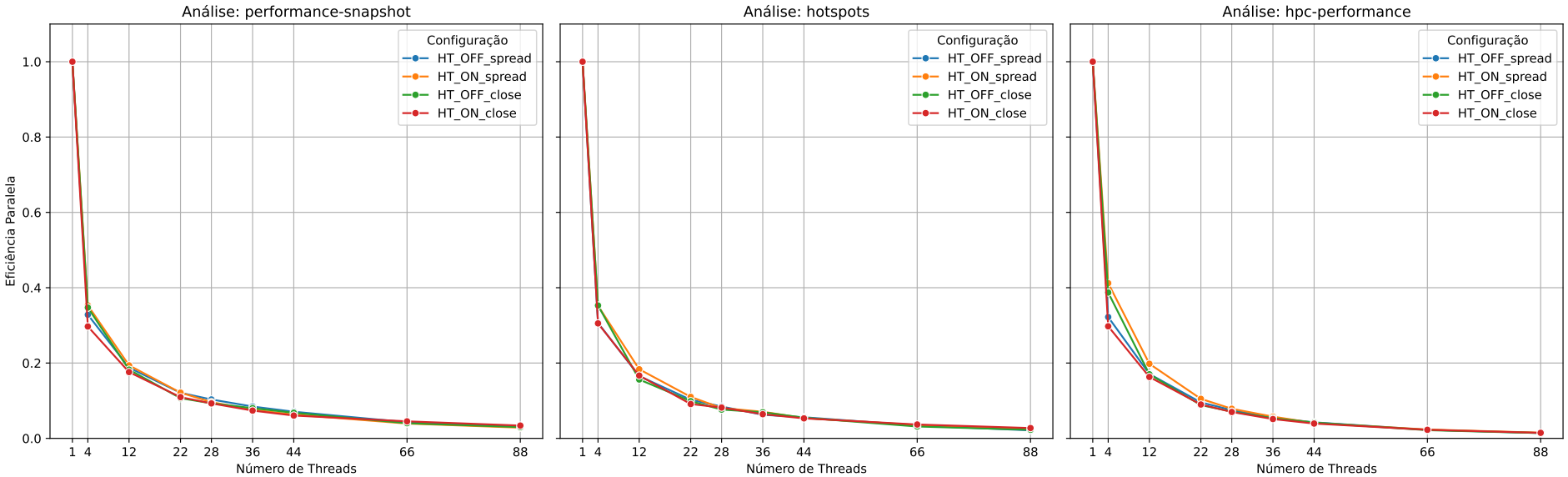
- E_p é a eficiência paralela.
- T_s é o tempo de execução sequencial (com 1 thread).
- N é o número de threads utilizados.
- T_p é o tempo de execução paralelo com N threads.

Os gráficos a seguir estão organizados similarmente aos últimos.

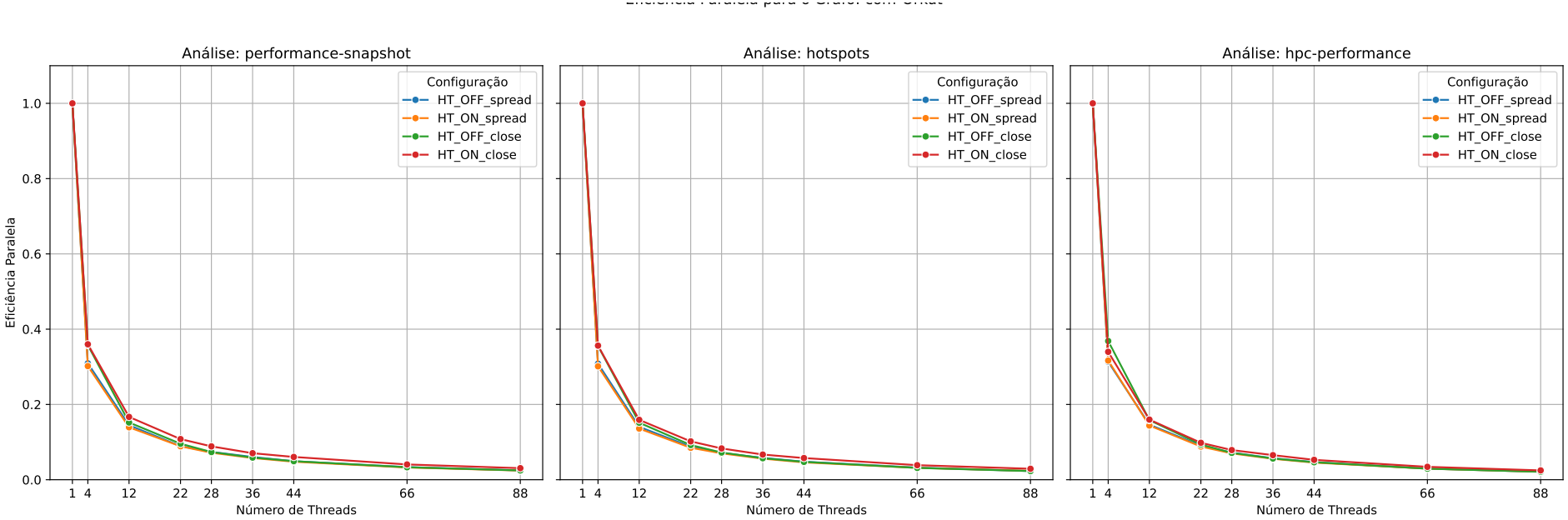
EFICIÊNCIA PARALELA - FRIENDSTER



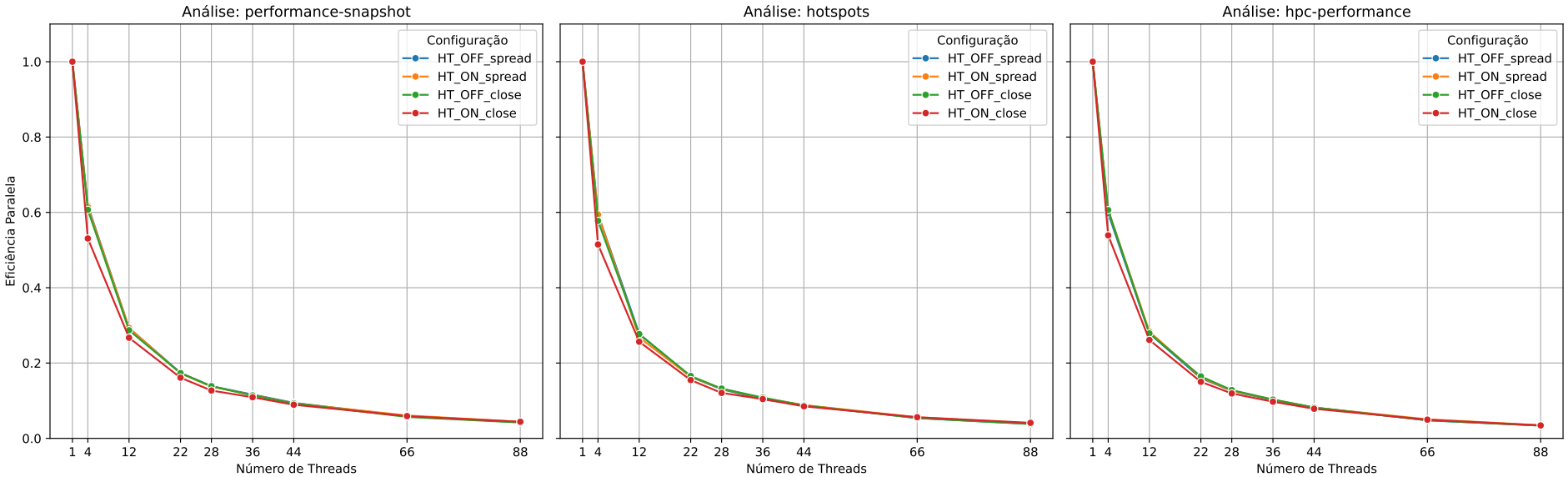
EFICIÊNCIA PARALELA - LIVEJOURNAL



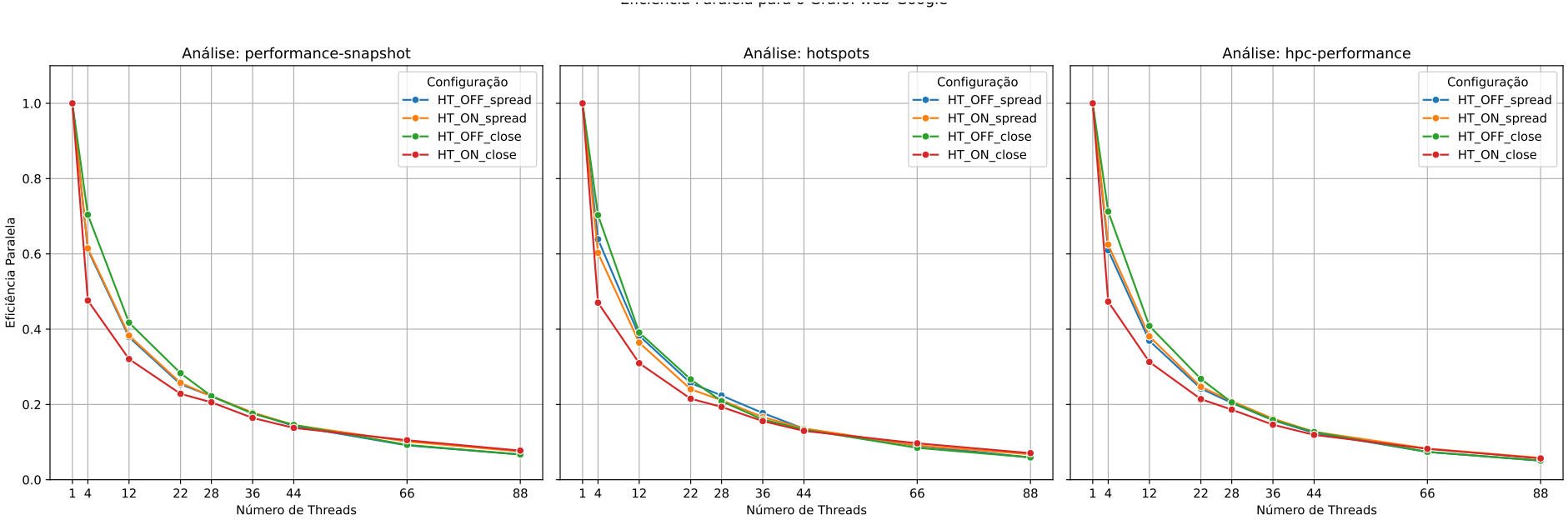
EFICIÊNCIA PARALELA - ORKUT



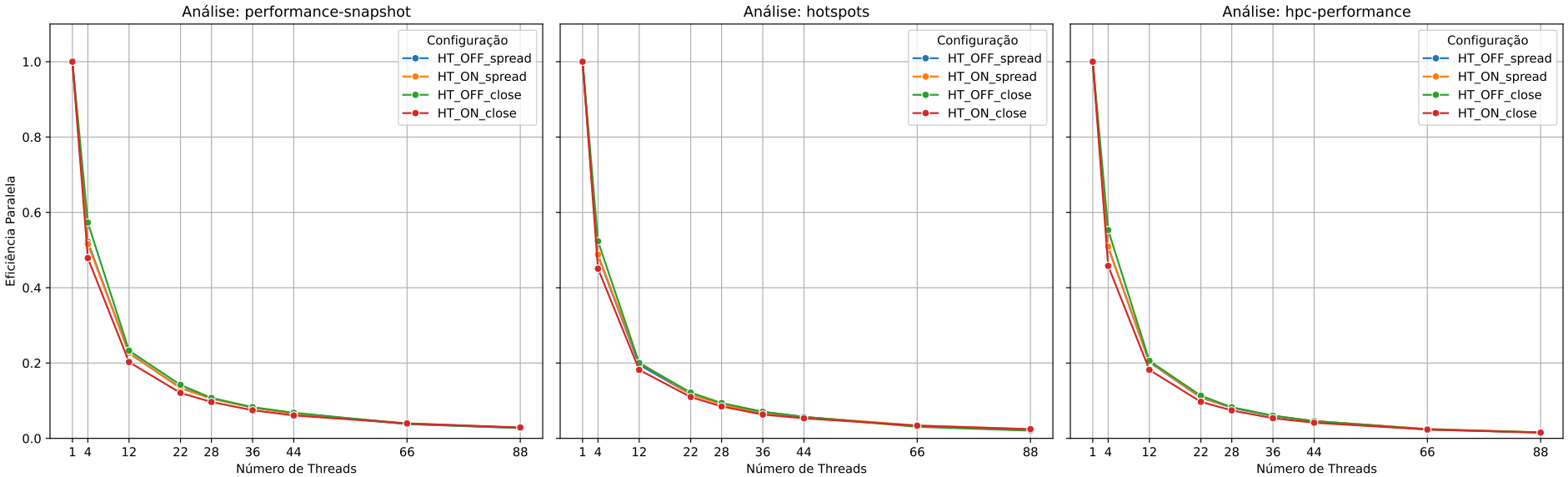
EFICIÊNCIA PARALELA - BERKSTAN



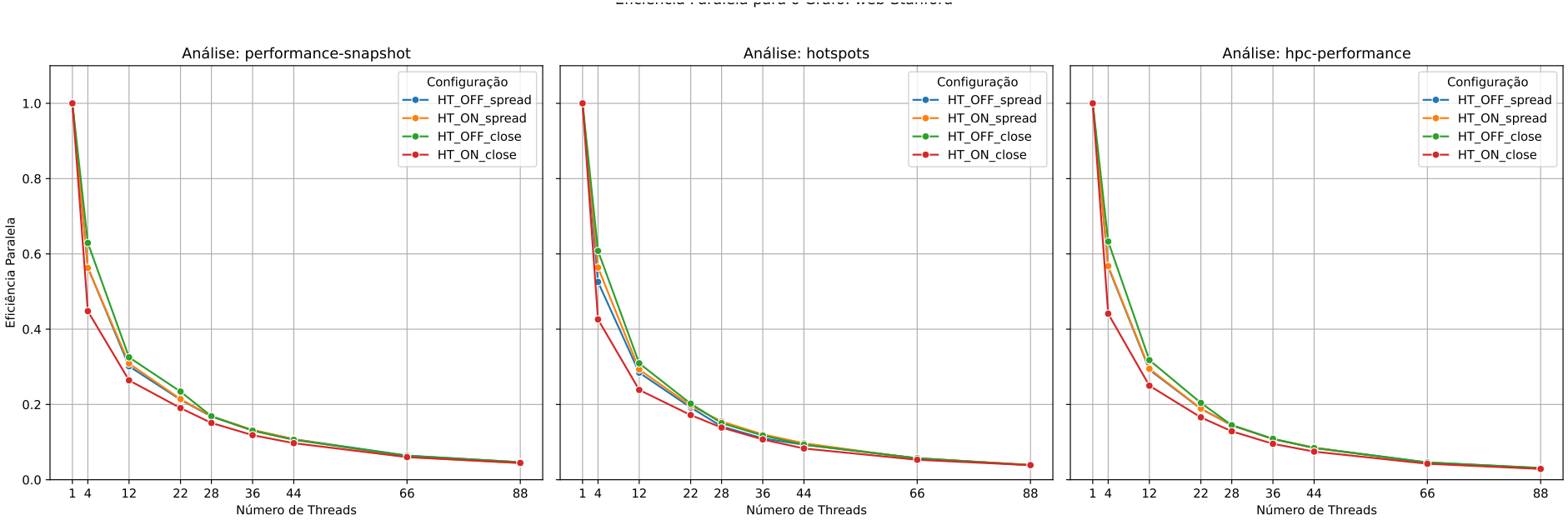
EFICIÊNCIA PARALELA - GOOGLE



EFICIÊNCIA PARALELA - NOTREDAME



EFICIÊNCIA PARALELA - STANFORD



DIFICULDADES ENFRENTADAS

- Vtune gerando muitos dados, a grande maioria não eram de interesse do grupo.
- Falta de disponibilidade da máquina.
  (Obrigado bmmoreira!)
- Conseguir deixar executando sem o Hyperthreading.
- Vtune gerando arquivos de saída mal formatados e grandes.

Mesmo com essa análise superficial, já conseguimos chegar a algumas conclusões:

- A política de binding de threads "close"consegue aproveitar muito bem a localidade de dados com muitas threads, mas com poucas threads essa eficiência não se reproduz.
- A presença de Hyperthreading produz diferenças significativas no speedup das aplicações, mas também houve casos onde estar com ele desligado tornava o código mais rápido.
- E o mais importante: **Não há configuração global pra nenhuma das entradas que cause o maior speedup!**

1. Começar a trabalhar no documento final de entrega do projeto (com o *template* em LaTeXprovidenciado pelo professor).
2. Começar a analisar o resto dos dados providenciados pelo Intel VTune, como uso dos núcleos físicos e lógicos, frequência média do processador, se é a memória que limita o desempenho, etc.
3. Finalmente, completar nosso *Notebook* utilizado com novos gráficos e novas análises.

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

OBRIGADO!

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