

Apresentação TCC

Investigação de técnicas de otimização
para algoritmos de aprendizado de
máquina

Orientador: Prof. Dr. João Paulo Papa

Co-orientador: Prof. Dr. Alexandro José Baldassin

Aluno: André Libório

andre.lb.ferraz@unesp.br



Sumário

1. Introdução
2. Fundamentação Teórica
3. Ferramentas
4. Vetorização
5. OPF
6. Galois
7. Conclusão



Introdução

Algoritmos de aprendizado de máquina demandam bastante poder computacional

Para tornar essas tecnologias mais acessíveis, este trabalho busca melhorar o desempenho por meio do uso de tecnologias de vetorização e paralelismo



Fundamentação Teórica

Paralelização

Principal métrica para comparar desempenho entre processadores de mesmo período nas últimas décadas é seu número de núcleos

São necessárias otimizações no código para tirar vantagem dos múltiplos núcleos

Principais dificuldades:

- Balanceamento de cargas entre as *threads*
- Sincronização e controle de concorrência

Para a paralelização, geralmente são utilizados facilitadores, como a API OpenMP

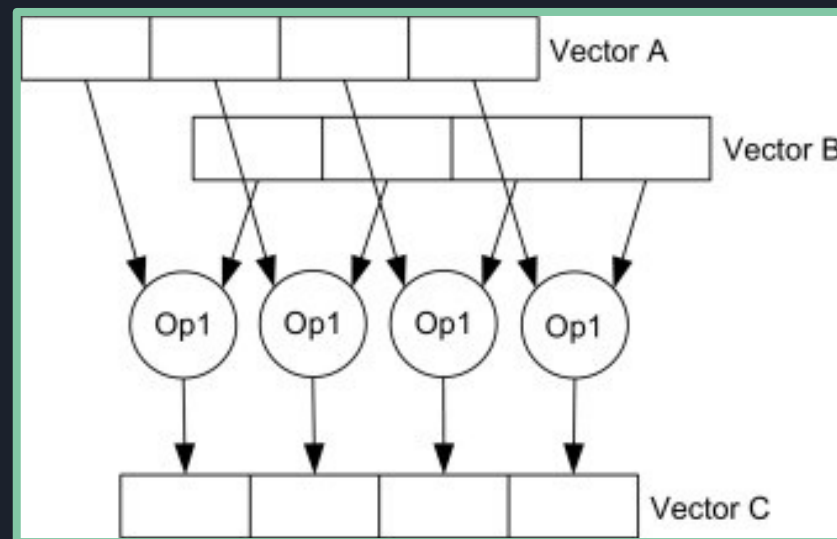
Fundamentação Teórica

Vetorização

Vetorização ou instruções SIMD (*Single Instruction Multiple Data*)

Aqui são utilizadas as recentes tecnologias AVX*:

- AVX2 (256 bits)
- AVX-512 (512 bits)



Fonte: Science Direct - Single Instruction Multiple Data¹

*Advanced Vector Extensions

¹ Disponível em: <<https://www.sciencedirect.com/topics/computer-science/single-instruction-multiple-data>>. Acesso em: 11 mai. 2022.



Fundamentação Teórica

Aprendizado de Máquina

Se beneficia dos avanços tecnológicos, viabilizando algoritmos mais complexos

Destaque para algoritmos baseados em grafos, como o OPF (*Optimum-Path Forest*), que são capazes de resolver problemas que visam a classificação dos elementos por meio de classes não separáveis com formas arbitrárias

Aqui será utilizado como base para o trabalho o *Parallel* OPF (POPF), vindo do trabalho realizado por Culquicondor et al. (2020)



Ferramentas

- Bash Script
 - Linguagem do terminal Linux
 - Utilizada para automação de processos
 - Execução
 - Tratamento de resultados
- Linguagem C
 - Permite otimizações em nível de *hardware*
 - Vetorização utilizando AVX2 e AVX-512 (Intel *Intrinsics*)
 - Uso de APIs para paralelização como o OpenMP
- Linguagem Python
 - Produção de gráficos utilizando o *matplotlib*



Metodologia

Sistema principal utilizado:

- 2x Intel Xeon Gold 5220 (36c/72t)
- Sistema Operacional CentOS 7.7
- GCC 7.3.1



Vetorização

Suporte AVX-512

Processadores Intel x86 possuem suporte para AVX-512

- Presente desde o lançamento nas linhas dedicadas a servidores (2016)
- Retirou suporte na última revisão da 12ª geração Intel Core

Processadores AMD x86 até então só possuíam suporte para AVX2

- Confirmado que a partir de sua próxima arquitetura Zen 4, haverá suporte para a tecnologia em todas suas linhas de processadores



Vetorização

Implementação

Antes de realizar a implementação no OPF:

- Verificar o desempenho da tecnologia
- Conferir o funcionamento e resultados

Para isso, fora utilizado o cálculo de multiplicação matricial pois possui:

- Código simples, portanto diagnóstico simples
- Alta capacidade de paralelização (OpenMP) e vetorização (AVX)



Vetorização

Implementação

Utilizando matrizes de dimensões 4000x4000:

- O uso matriz transposta no cálculo melhora o tempo de execução e *speedup*
 - 72t AVX-512 vs 1t AVX2 = 76,22x → 104,40x
 - 72t AVX-512 vs 1t OpenMP = 115,45x → 142,82x

Apresentou resultados corretos e bom desempenho!

Um artigo foi desenvolvido com esse trecho do trabalho e foi apresentado no *workshop* de iniciação científica ERAD-SP 2022 em abril

de 2022



OPF (*Optimum-Path Forest*)

Introdução

Algoritmo de aprendizado de máquina baseado em grafos

- Os nodos de um grafo são representados por um vetor de característica

Para calcular a semelhança entre os nodos, uma função para cálculo da distância entre cada par é utilizada, neste caso, o cálculo da distância Euclidiana

Esse cálculo demanda um grande tempo de computação

- Pode ser interessante realizar sua otimização!



OPF (*Optimum-Path Forest*)

Introdução

Equação do cálculo da distância Euclidiana utilizado:

$$d(u, v) = D * \log(1 + \|u - v\|^2) = D * \log(1 + \sum_{n=1}^f (u_i - v_i)^2)$$

onde D é uma constante, f o número de características, e u_i e v_i representam o i -ésimo componente dos vetores u e v , respectivamente

A somatória realizada no final da equação é uma operação realizada com vetores e pode se beneficiar da vetorização!



OPF (*Optimum-Path Forest*)

Experimentos

Antes de iniciar a implementação no POPF, foi criada uma versão isolada para verificar a corretude dos resultados

- Sem vetorização
- AVX2
- AVX-512

```
1  inicio
2      //carrega os dados do vetor1 e vetor2 em respectivos vetores AVX-512
3      vetor1avx = _mm512_load_pd(&vetor1[i]);
4      vetor2avx = _mm512_load_pd(&vetor2[i]);
5
6      //realiza as operacoes de subtracao, multiplicacao e incremento
       final
7      vetorAux = _mm512_sub_pd(vetor1avx, vetor2avx);
8      vetormul = _mm512_mul_pd(vetorAux, vetorAux);
9      vetFinalAvx = _mm512_add_pd(vetormul, vetormul);
10
11     //armazena os valores finais presentes no vetor AVX-512 vetFinalAvx
       em vetorFinal
12     _mm512_storeu_pd(&vetorFinal[i], vetFinalAvx);
13 fim
```

Pseudo-código utilizando AVX-512







OPF (*Optimum-Path Forest*)

Intrinsics

Intrinsic utilizada para carregar 512-bits da memória para um vetor AVX-512:

`_mm512_load_pd(&vetor[i]);`

			
Tamanho do vetor AVX utilizado	Instrução	Tipo da variável	Endereço do dato a ser acessado

Outros exemplos de *Intrinsics* utilizadas nos experimentos:

`_mm512_sub_pd(vet1avx, vet2avx)` → subtração de vetores AVX-512

`_mm512_mul_pd(vetorAux, vetorAux)` → multiplicação de vetores de AVX-512

`_mm256_hadd_ps(vetAvx, vetAvx)` → soma horizontal de vetores AVX2

`_mm512_storeu_pd(&vetor[i], vetAvx)` → armazena de vetores AVX-512 na memória



OPF (*Optimum-Path Forest*)

Implementação

Para realizar a implementação no POPF, foram necessárias modificações, para manter a estrutura já existente de entrada/saída do código

Em AVX-512, são carregados 16 elementos por vez (*float* → $512/(4*8)=16$)

```
1  Entrada: vetor1, vetor2, qtdFeatureDataset
2  Saida: dist
3  inicio
4      for (i=0; i <= qtdFeatureDataset-16; i+=16){
5          //carregamento do trecho do vetor de 16 elementos
6          vetorAvx1 = _mm512_load_ps(&vetor1[i]);
7          vetorAvx2 = _mm512_load_ps(&vetor2[i]);
8
9          //calcula da distancia euclidiana
10         vetAccAvx = vetAccAvx + ((vetorAvx1 - vetorAvx2) * (vetorAvx1 -
            vetorAvx2));
11     }
```





OPF (*Optimum-Path Forest*)

Implementação

```
12 // processo de reducao e armazenamento do resultado
13 __m256 low = _mm512_castps512_ps256(vetAccAvx);
14 __m256 high = _mm256_castpd_ps(_mm512_extractf64x4_pd (
    _mm512_castps_pd(vetAccAvx), 1));
15 vetAvxFinal = _mm256_add_ps(low, high);
16 __m256 tmp = _mm256_permute2f128_ps(vetAvxFinal, vetAvxFinal, 0x1);
17 vetAvxFinal = _mm256_add_ps(vetAvxFinal, tmp);
18 vetAvxFinal = _mm256_hadd_ps(vetAvxFinal, vetAvxFinal);
19 vetAvxFinal = _mm256_hadd_ps(vetAvxFinal, vetAvxFinal);
20
21 float tmp2[8] __attribute__((aligned(32)));
22 _mm256_store_ps(tmp2, vetAvxFinal);
23 dist += tmp2[0];
24 fim
```

Parte 2 da implementação otimizada no POPF



OPF (*Optimum-Path Forest*)

Implementação

O código anterior apresentou resultados corretos e com desempenho razoável

Mas ainda é possível reduzir o tempo de execução!

- Por meio de uma implementação híbrida
 - Utiliza AVX2 e AVX-512, de maneira a melhor utilizar a quantidade de *features*, ou tamanho do vetor de características do *dataset*
 - AVX-512 → 16 ou mais elementos *float*
 - AVX2 → 8 ou mais elementos *float*



OPF (*Optimum-Path Forest*)

Implementação

Os *datasets* utilizados são advindos do trabalho de Culquicondor et al. (2020)

Dataset	Instâncias	Características
MiniBooNE	130.064	50
SDD	58.509	48
Letter	20.000	16

Dentre os apresentados, apenas os 3 *datasets* abaixo são de interesse:

- **Características ≥ 16** \Rightarrow pode se beneficiar da vetorização com AVX-

512

$$(featuresDataset * tamanhoDado) / tamanhoVetorAVXEmBytes \geq 1$$



OPF (*Optimum-Path Forest*)

Resultados

Dados obtidos utilizando os 3 *datasets* com apenas 1 *thread* (em segundos):

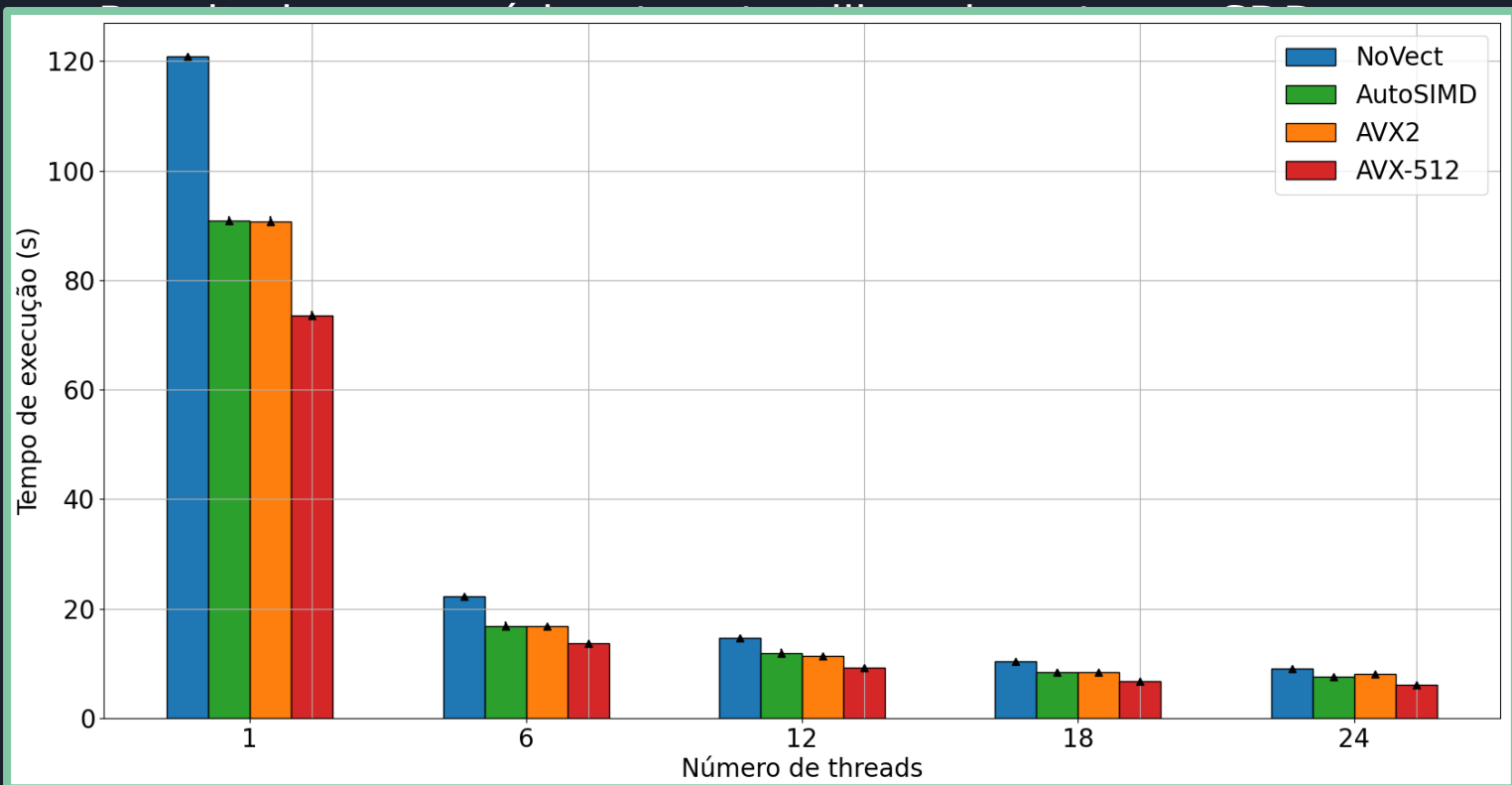
Dataset	Caract.	NoVect	AVX2	AVX-512	NoVect/AVX-512	AVX2/AVX-512
MiniBooNE	50	1348,50	1011,60	927,68	45,36%	9,05%
SDD	48	120,84	90,78	73,49	64,43%	23,53%
Letter	16	15,75	16,15	13,10	20,23%	23,28%

Resultados:

- Apresentam ganhos generalizados
- Destaque para o SDD
 - 64,43% de ganho sobre NoVect
 - 23,53% de ganho sobre AVX2

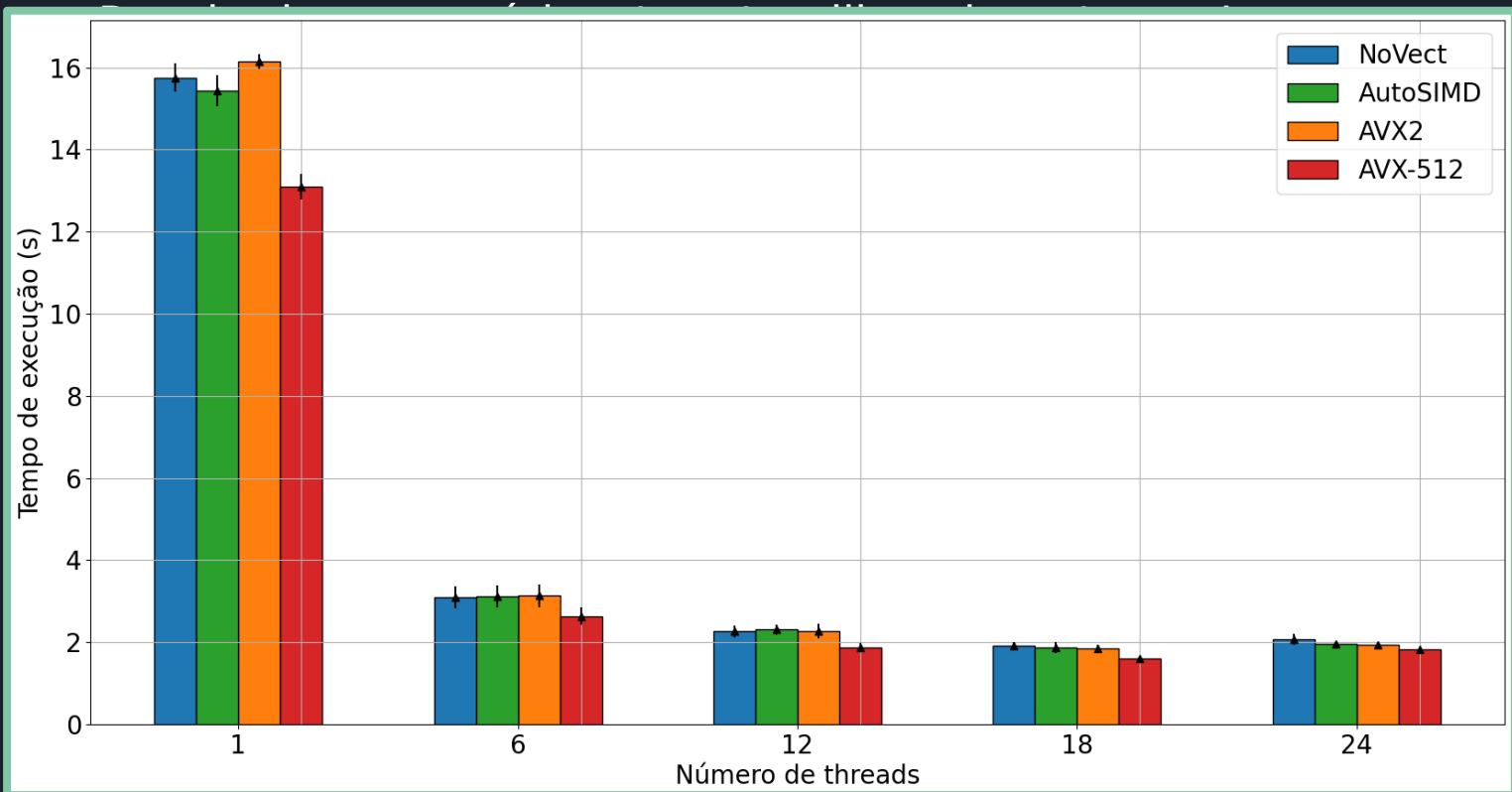
OPF (*Optimum-Path Forest*)

Resultados



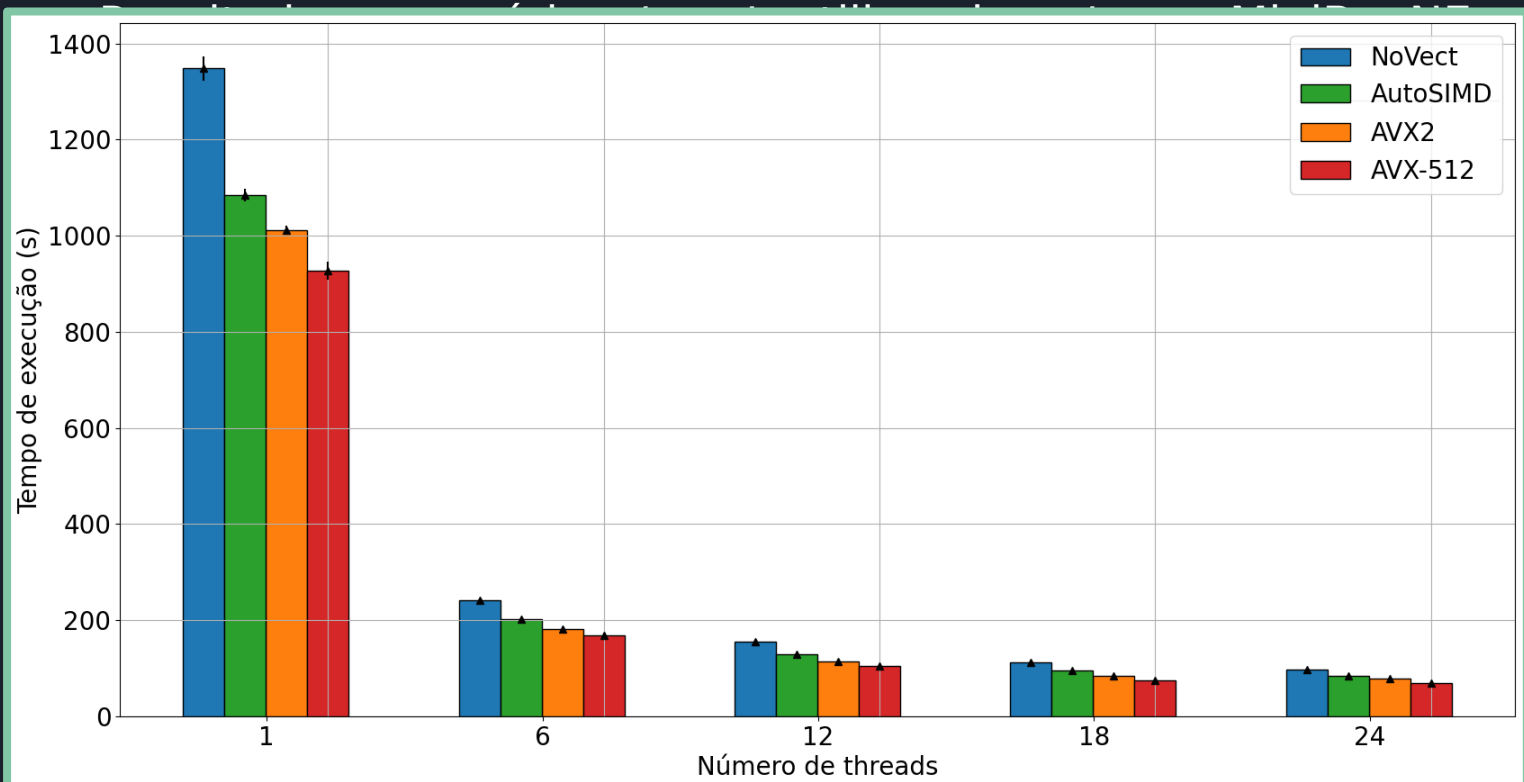
OPF (*Optimum-Path Forest*)

Resultados



OPF (*Optimum-Path Forest*)

Resultados





OPF (*Optimum-Path Forest*)

Análise dos Resultados

Os resultados com AVX-512 são positivos!

Ganho com a vetorização se torna menos relevante conforme o número de *threads* aumenta

- Paralelização apresenta um bom ganho, reduzindo impacto da vetorização

Também apresenta um bom desempenho com auto-vetorização do OpenMP



OPF (*Optimum-Path Forest*)

Confirmação de Desempenho

Seria esse comportamento presente apenas na máquina utilizada para testes?

Para isso, fora utilizado um *ultrabook* com suporte a AVX-512 para realizar os mesmos testes:

- Intel Core i7 1065G7 (4c/8t)
- Sistema Operacional Pop_OS 22.04
- GCC 11.2.0



OPF (*Optimum-Path Forest*)

Confirmação de Desempenho

Dados obtidos utilizando os 3 *datasets* com apenas 1 *thread* (em segundos):

Dataset	Caract.	NoVect	AVX2	AVX-512	NoVect/AVX-512	AVX2/AVX-512
MiniBooNE	50	861,63	640,05	611,26	40,96%	4,71%
SDD	48	56,23	32,52	26,42	112,83%	23,09%
Letter	16	6,03	6,38	5,07	18,93%	25,84%

Resultados apresentam ganhos distintos, mas ainda apresentando ganhos!

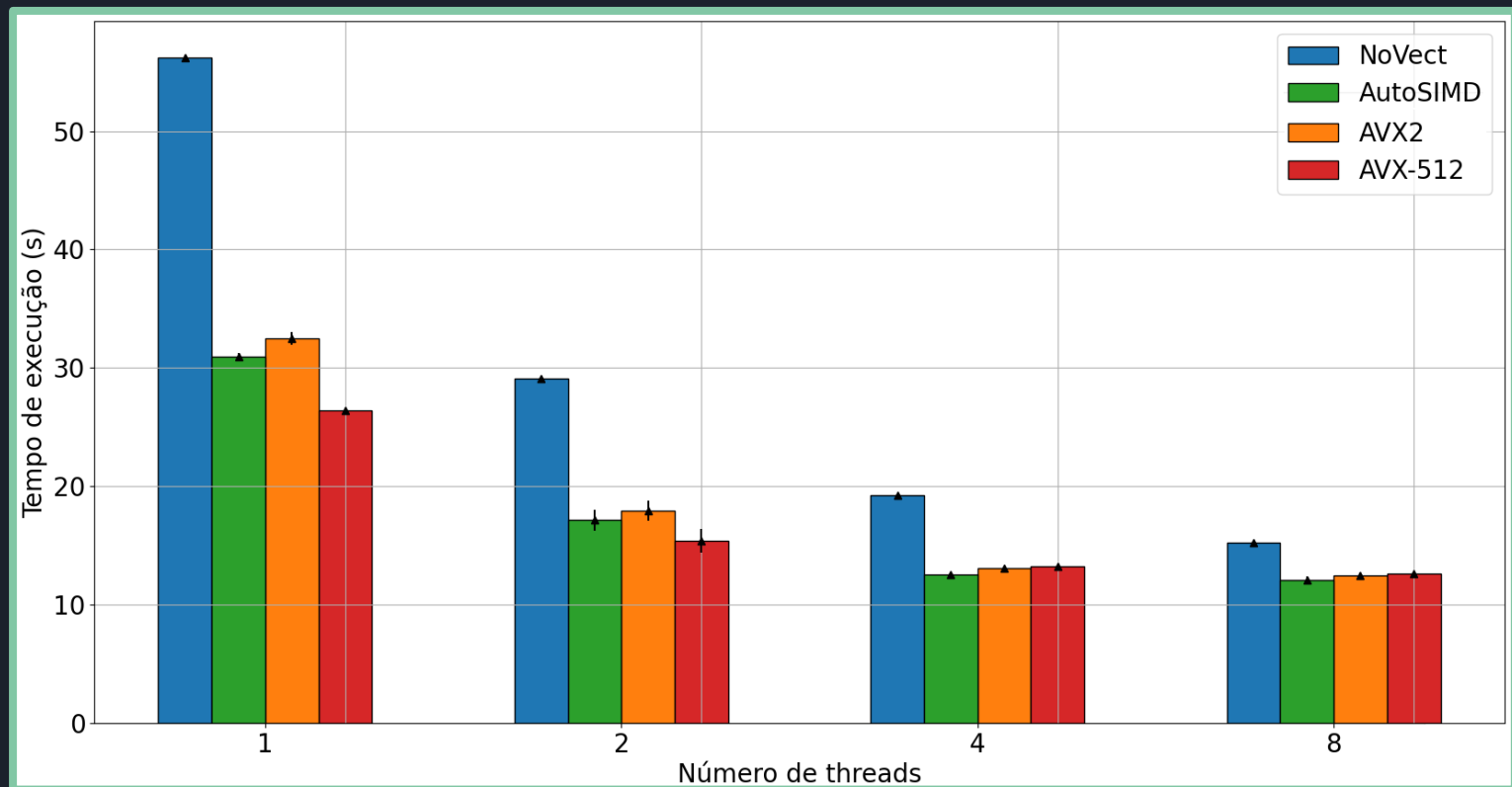
Tais discrepâncias podem se dar à diferente arquitetura do processador!

Resultados:

OPF (*Optimum-Path Forest*)

Confirmação de Desempenho

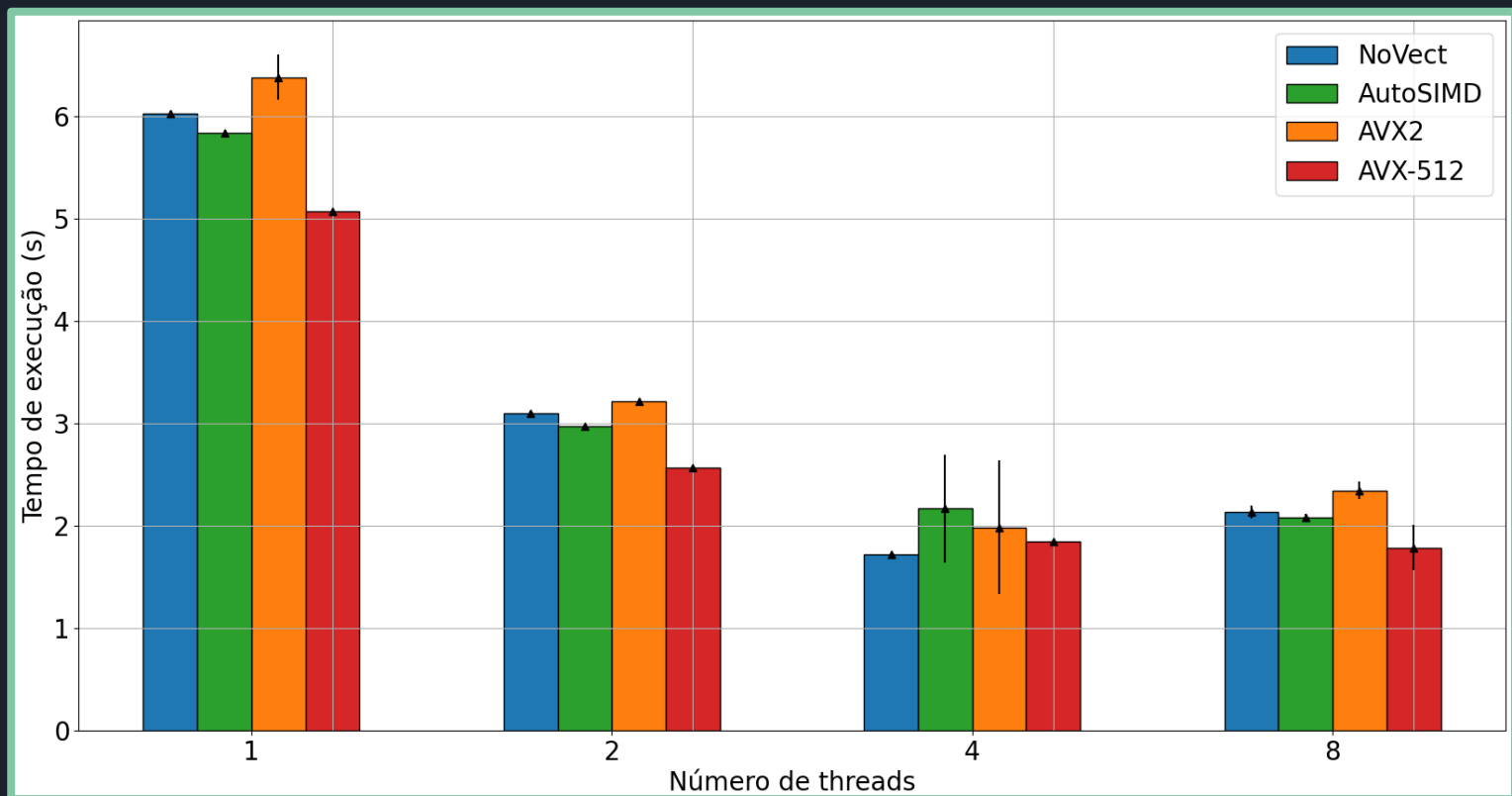
Resultados com vários *threads* utilizando o *dataset* SDD



OPF (*Optimum-Path Forest*)

Confirmação de Desempenho

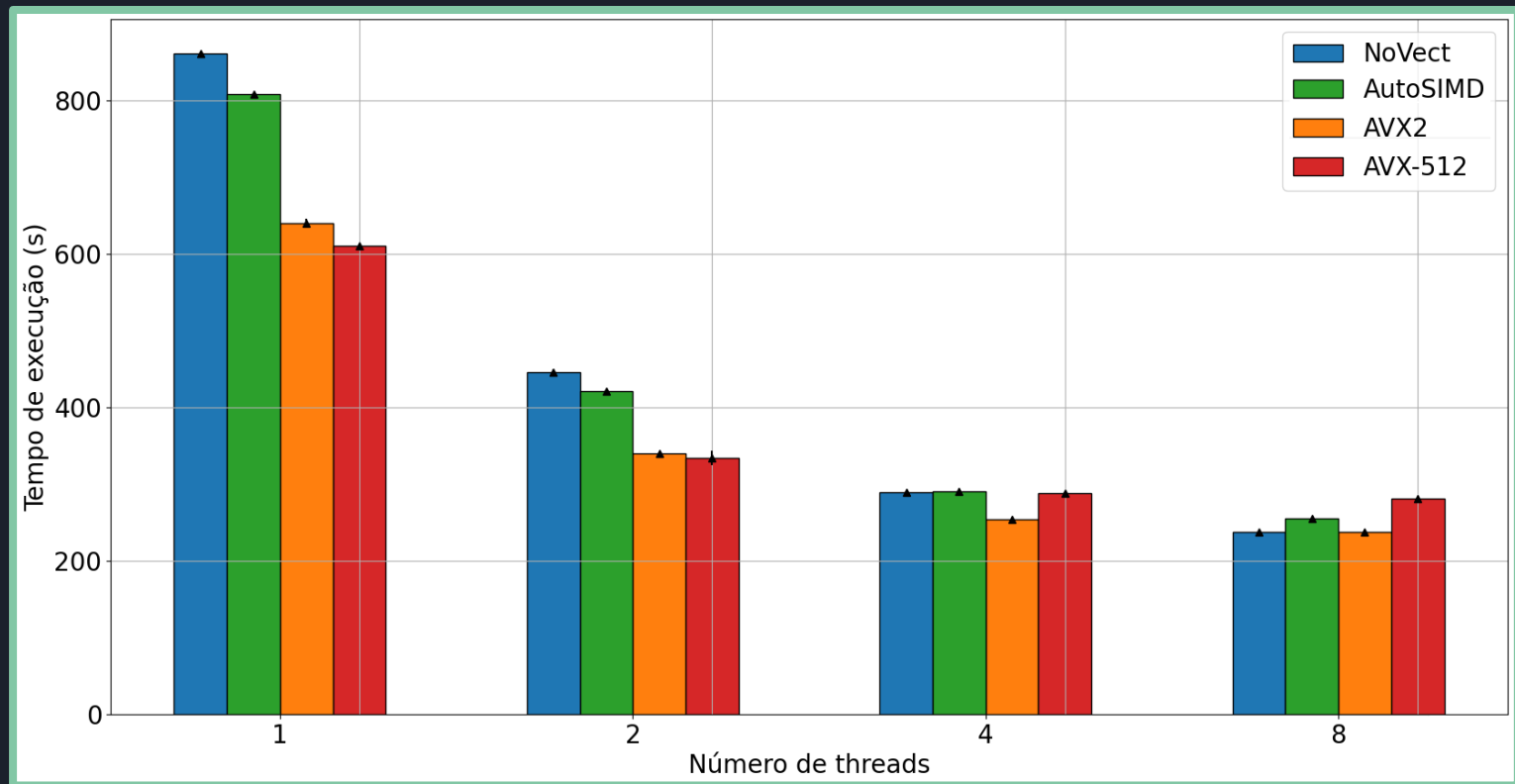
Resultados com vários *threads* utilizando o *dataset* Letter



OPF (*Optimum-Path Forest*)

Confirmação de Desempenho

Resultados com vários *threads* utilizando o *dataset* MiniBooNE





Galois

Introdução

É um arcabouço computacional que permite explorar o paralelismo de dados amorfos em algoritmos irregulares na linguagem C++ sem a necessidade de uma programação paralela explícita

- Foi escolhido pelo seu bom desempenho em algoritmos baseados em grafos!
- Pelo funcionamento do OPF, essa implementação pode trazer benefícios!




Galois

Introdução

Como a maior parte do trabalho foi voltado para a vetorização:

- Não foi possível realizar toda a reimplementação do OPF
- Os testes aqui realizados consideram apenas a aplicação em MST* e não abordam Dijkstra
- O estudo teve o foco em avaliar a viabilidade de implementação no OPF

A verificação de desempenho foi feita utilizando o algoritmo de MST Boruvka



Galois

Metodologia

- Valores de tempo obtidos por meio do próprio Galois
- Dados são relativos a média de 3 execuções realizadas
 - Foram apresentados apenas os valores médios devido a baixa variabilidade
- Speedup foi obtido dividindo-se o tempo de execução sequencial de cada configuração pelos tempos com diferentes números de *threads*.

Foram utilizados *datasets* fornecidos como parte do repositório público do Galois

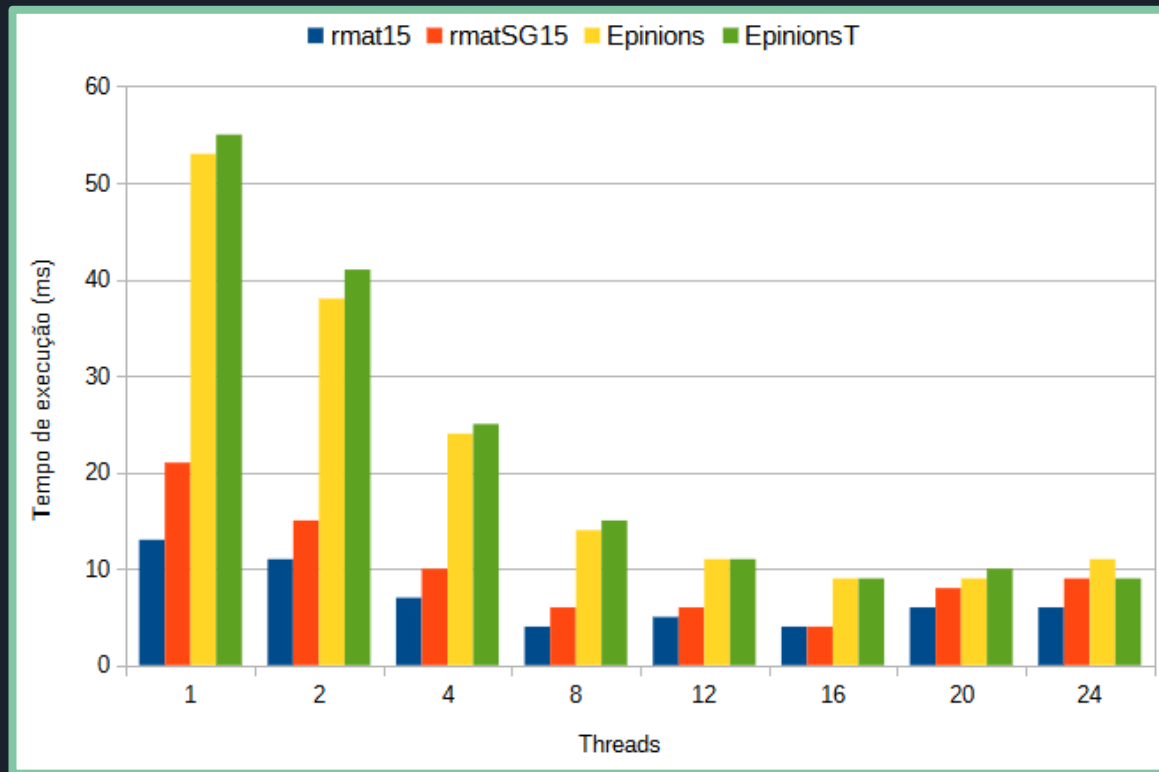
- R-MAT e Epinions

Galois

Resultados

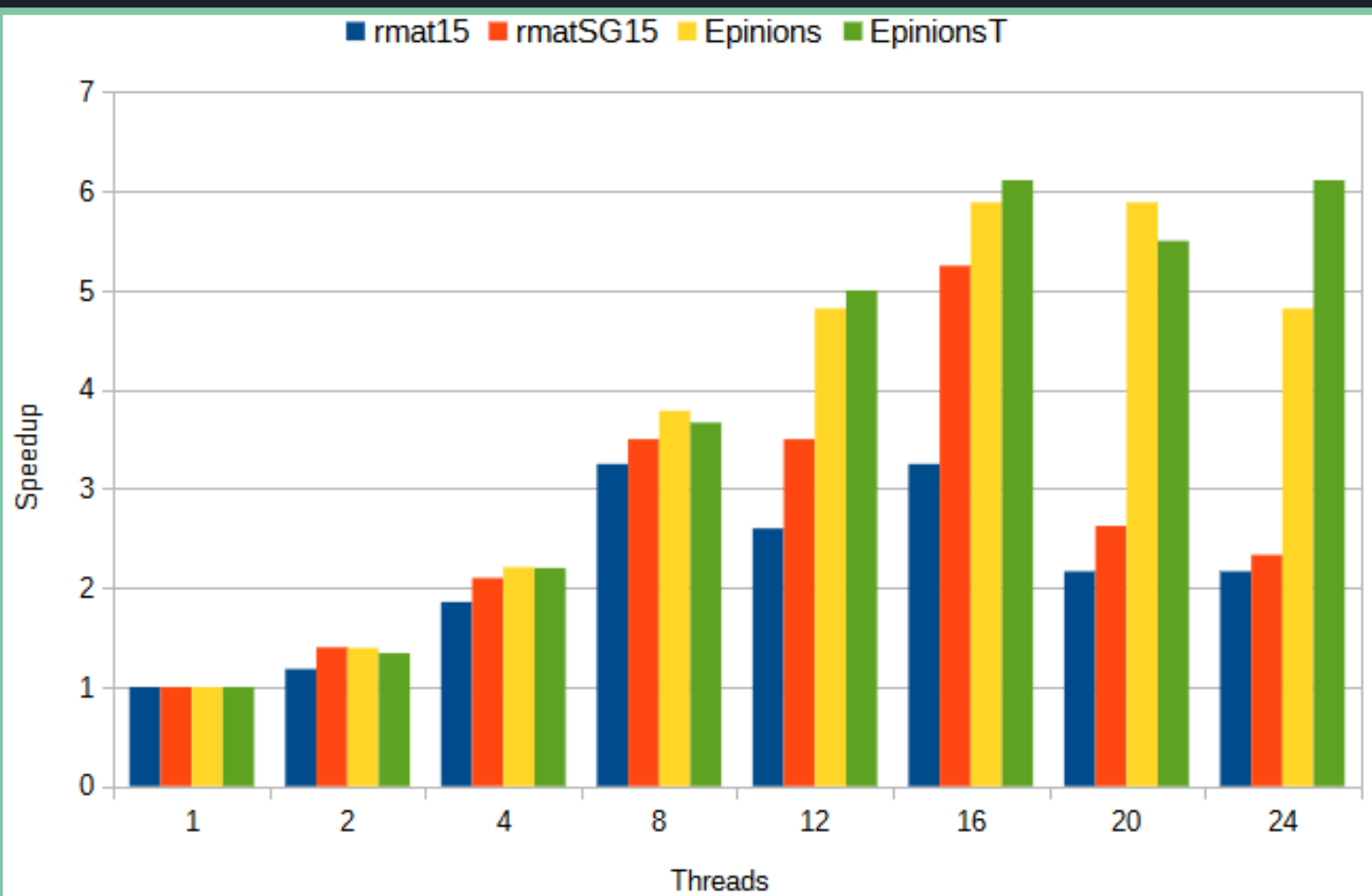
Apresenta uma melhora de desempenho com até 16 *threads*

Após essa marca, o desempenho tende a ter uma leve piora e estabilizar



Galois

Resultados



Galois

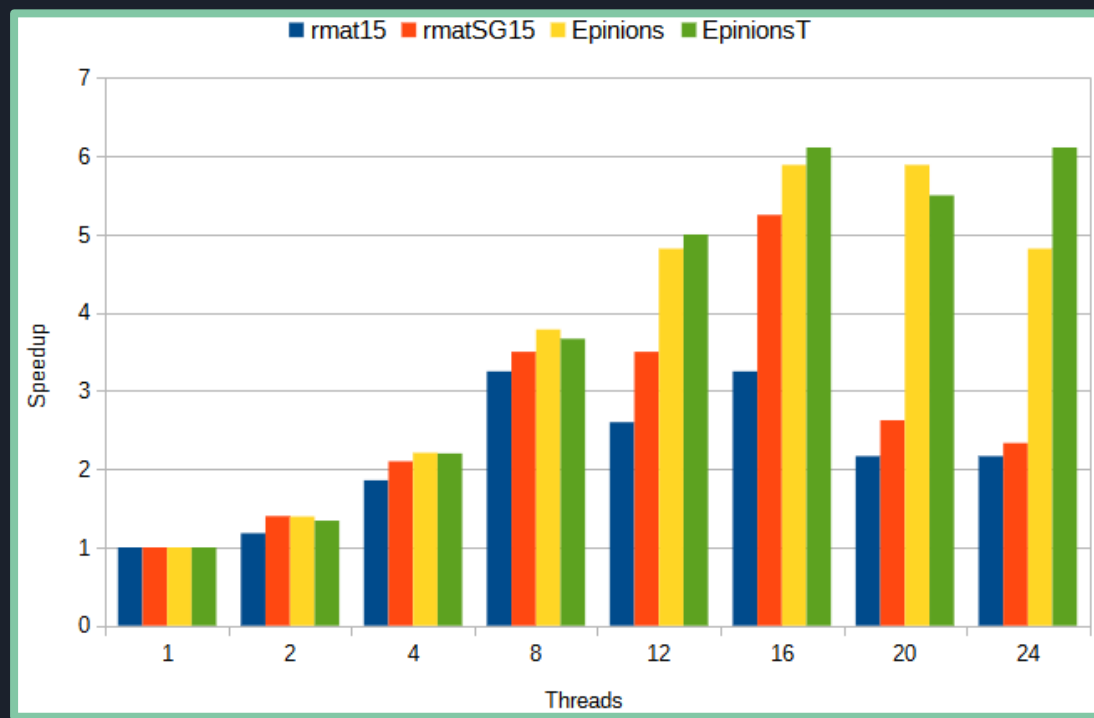
Resultados

Com até 8 *threads*:

- Comportamento esperado, mas com *speedup* global limitado entre 3x e 4x
 - Longe do 8x teórico

Ambos *datasets* atingem o maior desempenho com 16 *threads*:

- Epinions apresenta melhor desempenho (até 6x)
- R-MAT possui menor desempenho:
 - rmat15 com 3,25x
 - rmatSG15 com 5,25x





Galois

Resultados

Os dados apresentados, apesar das limitações, apresentam resultados promissores

Especialmente se considerarmos outros trabalhos da mesma natureza:

- Mariano et al. (2015) por exemplo:
 - Utiliza também um sistema Intel para os testes
 - Obtém bons resultados com o *dataset* road-network com Galois
 - *Speedup* de 6,06x com 8 *threads*

Uma possibilidade é que seu desempenho seja bastante dependente do *dataset*



Conclusão

A implementação de AVX-512 no POPF gerou bons resultados com 1 *thread*:

- Ganhos entre 20,23% e 64,43% sobre a implementação não vetorizada
- Ganhos entre 9,05% e 23,53% sobre a implementação com AVX2

E até mesmo na configuração *ultrabook* com 1 *thread*:

- Ganhos entre 19,93% e 112,83% sobre a implementação não vetorizada
- Ganhos entre 4,71% e 26,84% sobre a implementação com AVX2



Conclusão

É possível concluir que a implementação encontrou sucesso, com um ganho bastante significativo de desempenho global com a implementação AVX-512

Um artigo também fora submetido contendo os resultados aqui apresentados para o *workshop* de iniciação científica WSCAD-WIC 2022 que ocorrerá em outubro



Conclusão

Por fim, o estudo com Galois apresentou resultados razoáveis com até 16 *threads*

O estudo por Mariano et al. (2015) indica que é possível obter melhor desempenho

- Existe potencial para uma reimplementação no POPF!

Seria de interesse para trabalhos futuros realizar a implementação do Galois para outros algoritmos de grafos baseados no OPF e analisar os ganhos obtidos.



Referências Bibliográficas

CULQUICONDOR, A.; BALDASSIN, A.; CASTELO-FERNÁNDEZ, C.; CARVALHO, J. P. de; PAPA, J. P. An efficient parallel implementation for training supervised optimum-path forest classifiers. *Neurocomputing*, Elsevier, v. 393, p. 259–268, 2020.

MARIANO, A.; PROENCA, A.; SOUSA, C. D. S. A generic and highly efficient parallel variant of boruvka's algorithm. In: 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. [S.l.: s.n.], 2015. p. 610–617.

Obrigado pela atenção!