



Solução de um Problema de Equação de Burgers de Viscosidade Unidimensional Usando Rede Neural Informada por Física (PINN) e Método da Quadratura Gaussiana (GQM)

Eduardo Furlan

CAP-421 Aprendizado Profundo

2022-12-16

Por que PINN?

- Nova abordagem de ML para solução de PDE
- É possível obter modelos substitutos aproximadores
- Pode ser uma alternativa a métodos numéricos
- Pode aumentar a eficiência de modelos existentes
- Avaliar a aplicação em modelos meteorológicos (ex: módulo de radiação ecRad do MONAN)

1

Introdução

Objetivos deste trabalho

- Implementação de um caso de teste (equação de viscosidade 1D de Burgers)
- Uso de 2 abordagens, PINN (aprendizado de máquina) e GQM (método numérico)
- Comparar as eficiências das 2 abordagens

2

Abordagens e recursos

Apenas alguns recursos

- **TensorFlow: biblioteca de código aberto para ML e IA**
- **Fortran 90: usado no método numérico GQM**
- **OpenMP: biblioteca de paralelização por Threads**
- **JupyterLab: ambiente interativo e para a web**
- **SDumont: supercomputador do LNCC**

3

Caso de teste:

**Viscosidade 1D de
Burgers**

Caso de teste

- **Problema da equação de viscosidade 1D de Burgers**

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$$

- **Implantação 2 modelos matemáticos:**
 - **Rede Neural Informada por Física (PINN)**
 - **Método da Quadratura Gaussiana (GQM)**

Rede Neural Informada por Física (PINN)

- PINN embute PDE como parte da ANN
- Pode ser considerado um tipo de RL (reforço) que usa o gradiente como uma *função política*, e a PDE tem o papel de RL baseado em modelo
- Também pode ser considerada supervisionada, no caso de usar dados experimentais e PDEs que funcionam como supervisão

A PDE foi embutida na função de perda

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$$

$$u_t + u u_x - (0.01/\pi) u_{xx} = 0.$$

Condições iniciais (IC) e de contorno (BC):

$$u(0, x) = -\text{sen}(\pi x),$$

$$u(t, -1) = u(t, 1) = 0.$$

Exemplo simplificado de implementação

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0.$$

```
def net_u(self, x, t):  
    u = self.neural_net(tf.concat([x, t], 1), self.weights, self.biases)  
    return u  
  
def net_f(self, x, t):  
    u = self.net_u(x, t)  
    u_t = tf.gradients(u, t)[0]  
    u_x = tf.gradients(u, x)[0]  
    u_xx = tf.gradients(u_x, x)[0]  
    f = u_t + u * u_x - self.nu * u_xx  
    return f
```

TensorFlow

O treinamento minimiza a perda do MSE

Erro Médio
Quadrático

$$MSE = MSE_u + MSE_f,$$

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2,$$

Condição Inicial
(IC) e de
contorno (BC)

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2.$$

Pontos de
Colocação (CP)
aleatórios no
domínio da PDE

Parâmetros

- **Treinamento: 100 pontos IC+BC, e 10000 CP**
 - Aleatórios, dentro do domínio da PDE
- **Rede MLP com 10 camadas, sendo que 8 são escondidas com 20 neurônios cada**
- **Função de otimização: L-BFGS**
- **Função de ativação: tangente hiperbólica**

Podem ser usadas outras arquiteturas

- Apesar de ter sido usado MLP nesta implementação, na literatura também se encontra outras, como:
 - CNN (Convolucionais)
 - RNN (Recorrente)
 - AE (Auto-encoder)
 - DBN (Deep belief)
 - GAN (Generative adversarial)
 - BDL (Bayesian)

Método numérico implementado

- Método da Quadratura Gaussiana (GQM)
- Método numérico iterativo de aproximação
- Usa uma grade de 100 pontos tempo X 256 de espaço
- O resultado do GQM é usado para comparar com o resultado da PINN

4

Resultados: análise de desempenho

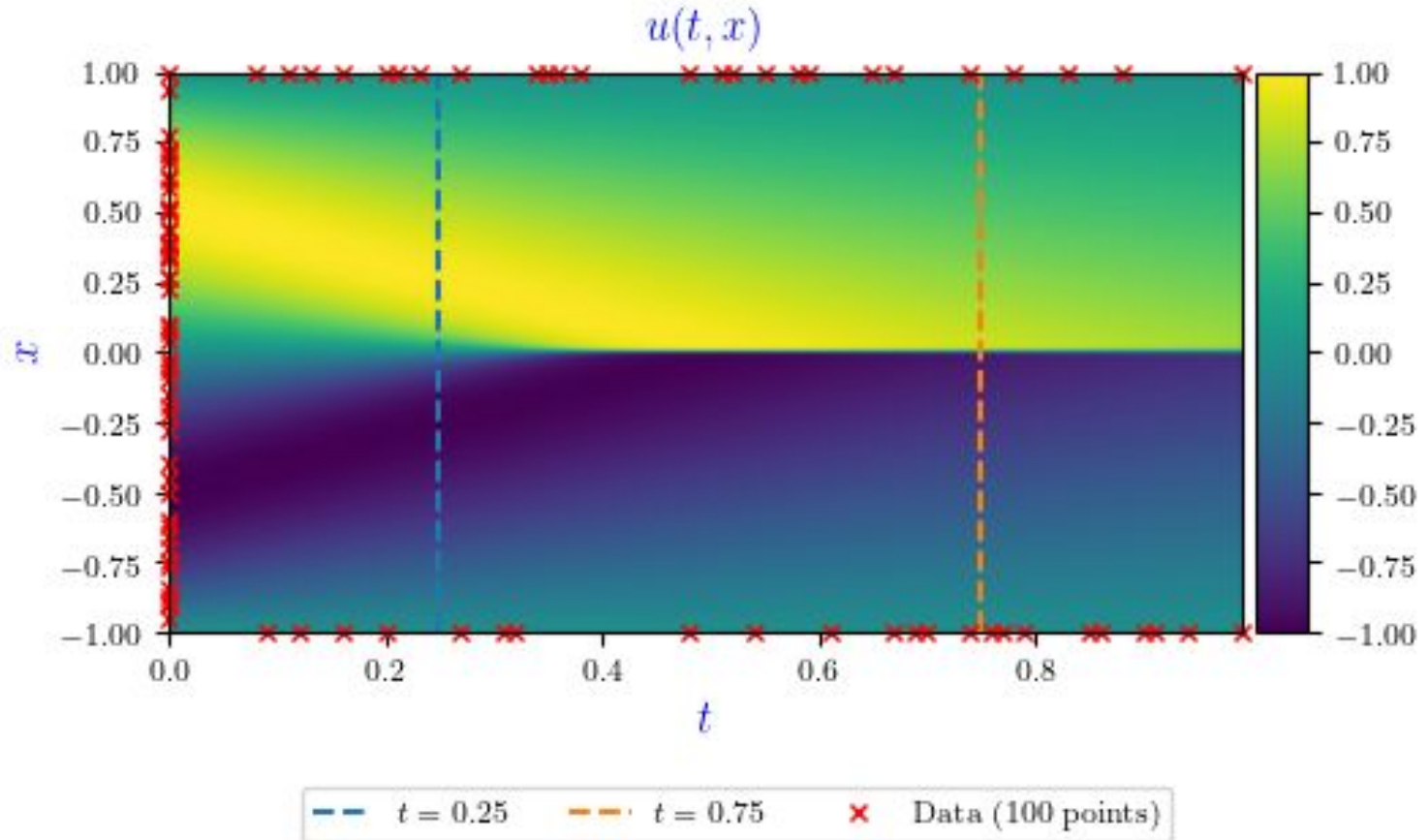
Ambiente (Santos Dumont LNCC)

- **Nó B710: 2x Xeon E5-2695v2 12-core**
- **Nó B715: 2x Xeon E5-2695v2 12-core + 2x Tesla K40**
- **Nó Sequana X: 2x Xeon 6152 22-core + 4x Volta V100**
- **GNU Fortran 4.8.5, OpenMP 3.1, Python 3.7, TensorFlow 1.15, e outros**
- **Execução paralela (PINN e GQM) usando Threads**

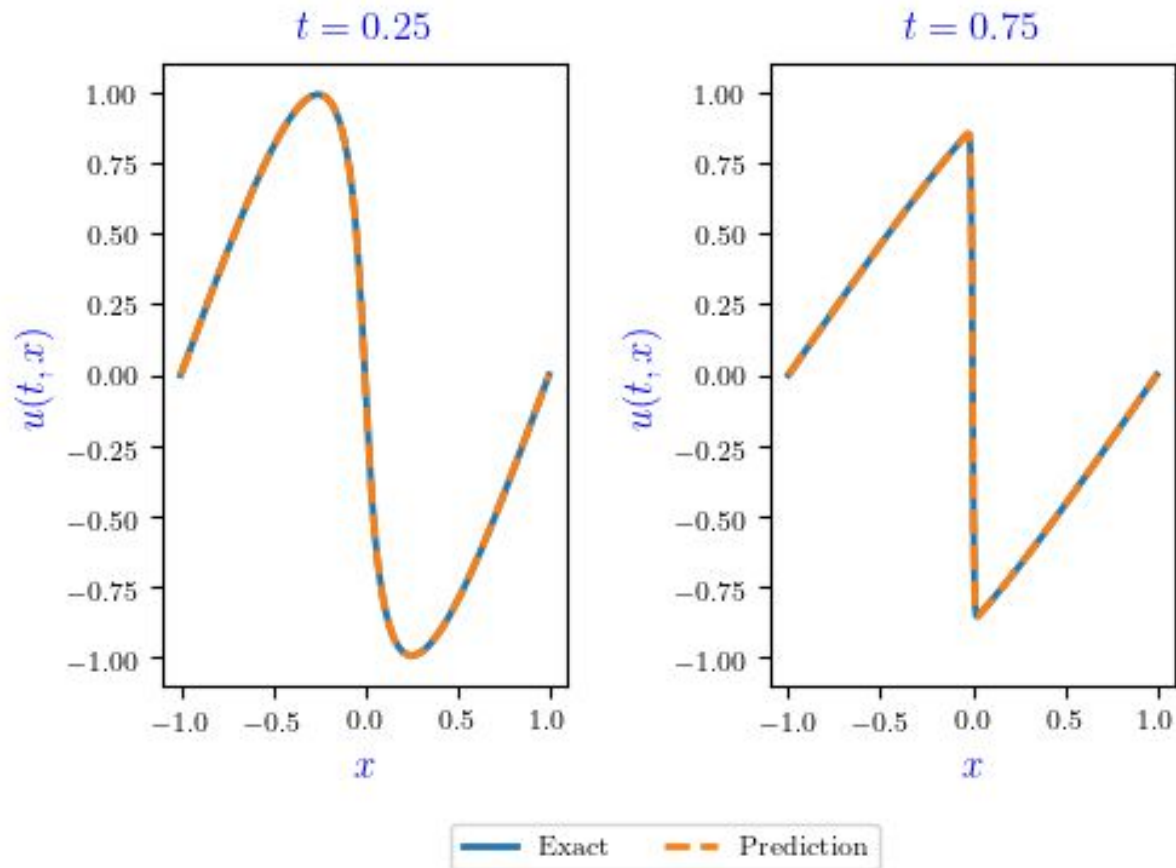
Medição

- **GQM:**
 - Como o tempo de execução é pequeno, a medição considera 1000 execuções
- **PINN:**
 - Predição: também 1000 execuções
 - Treinamento: executado 1 única vez

Resultado PINN



Instantâneos de tempo

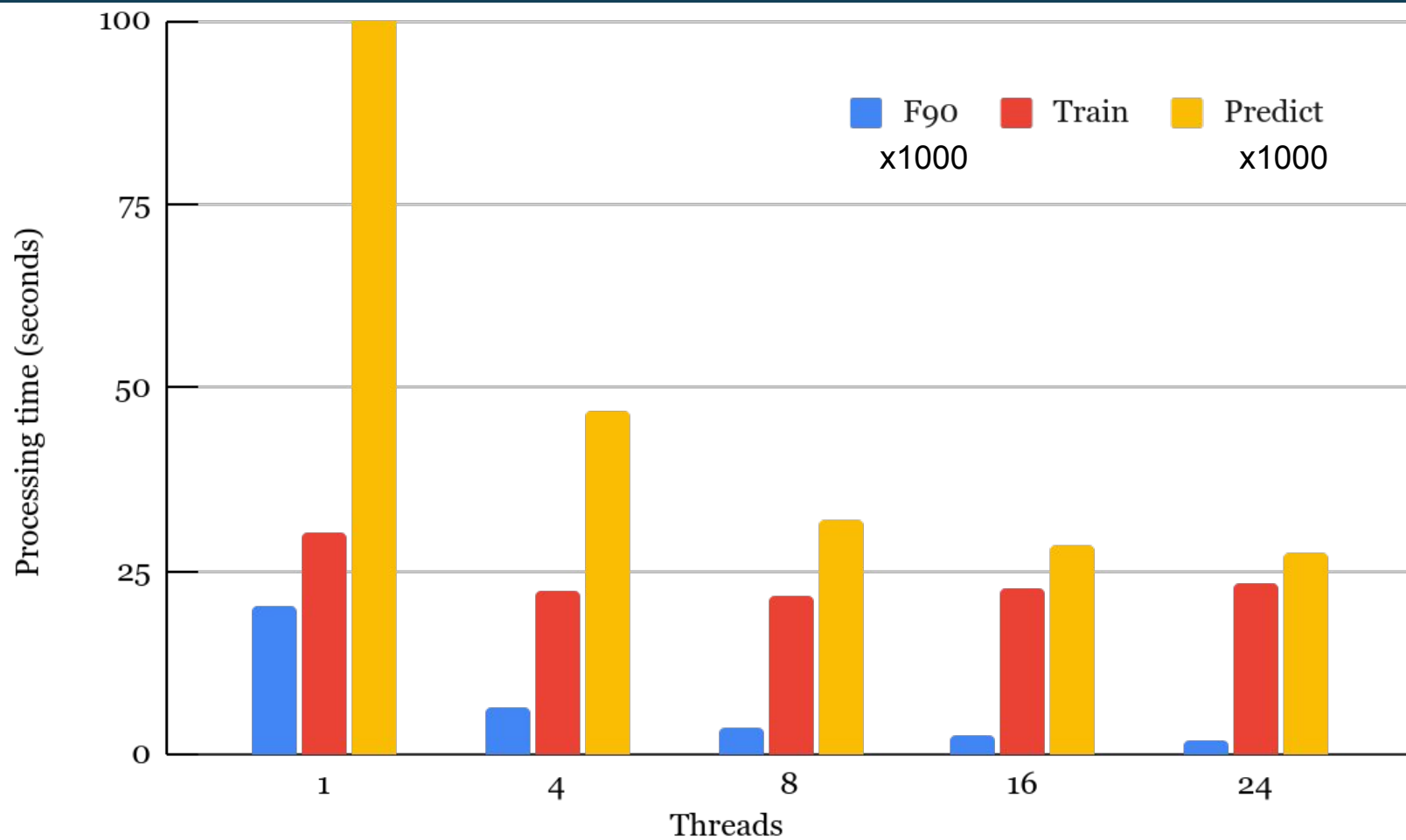


Tempos de processam.[s]/Speedup/Eficiência

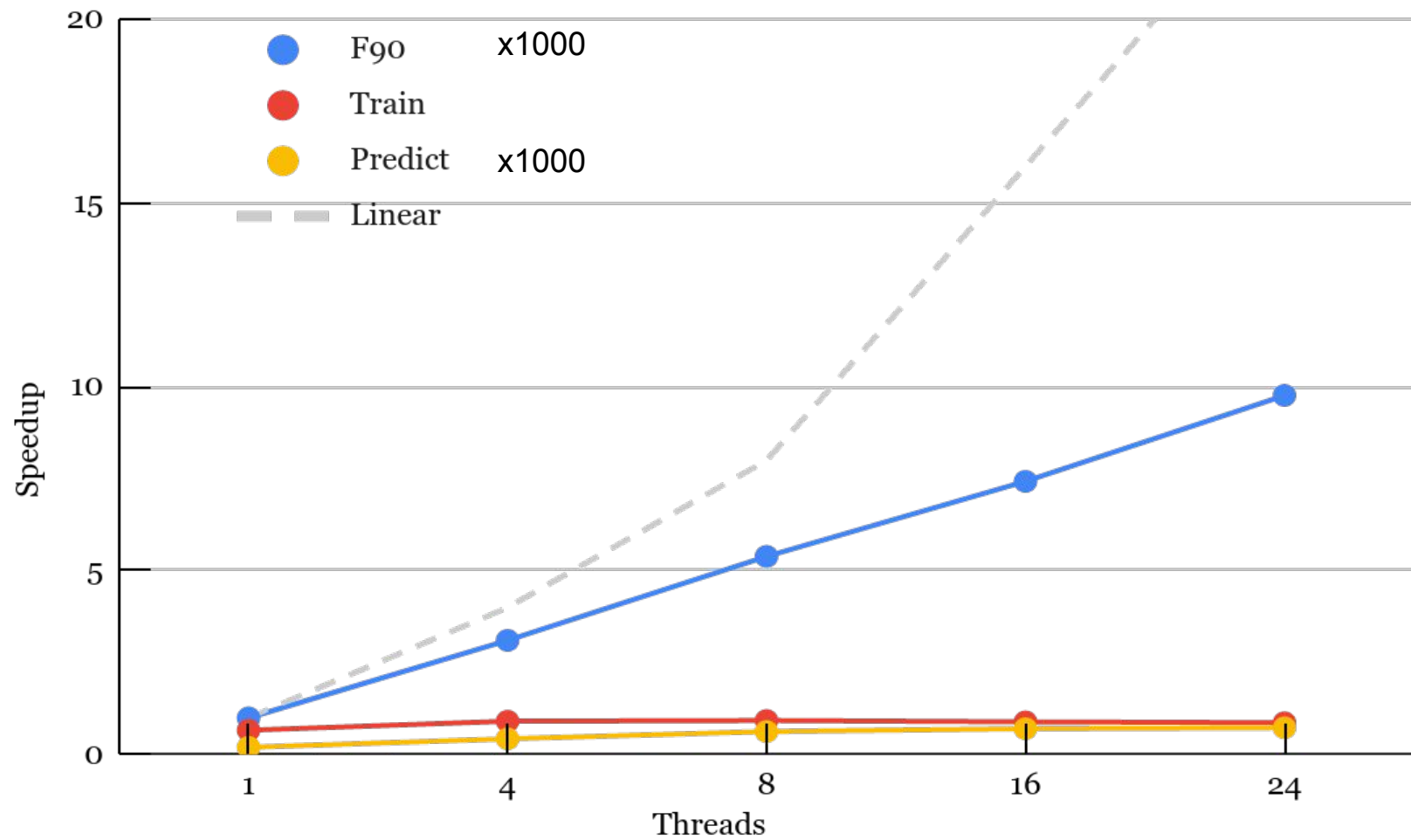
nós
B710

Profiling	Number of OpenMP threads				
	1	4	8	16	24
<i>Processing time (seconds)</i>					
F90	20.16	6.49	3.74	2.71	2.06
Train	30.33	22.14	21.69	22.56	23.21
Predict	100.64	46.77	32.06	28.52	27.57
<i>Speedup</i>					
F90	1.00	3.11	5.39	7.43	9.77
Train	0.66	0.91	0.93	0.89	0.87
Predict	0.20	0.43	0.63	0.71	0.73
<i>Parallel efficiency</i>					
F90	1.00	0.78	0.67	0.46	0.41
Train	0.66	0.23	0.12	0.06	0.04
Predict	0.20	0.11	0.08	0.04	0.03

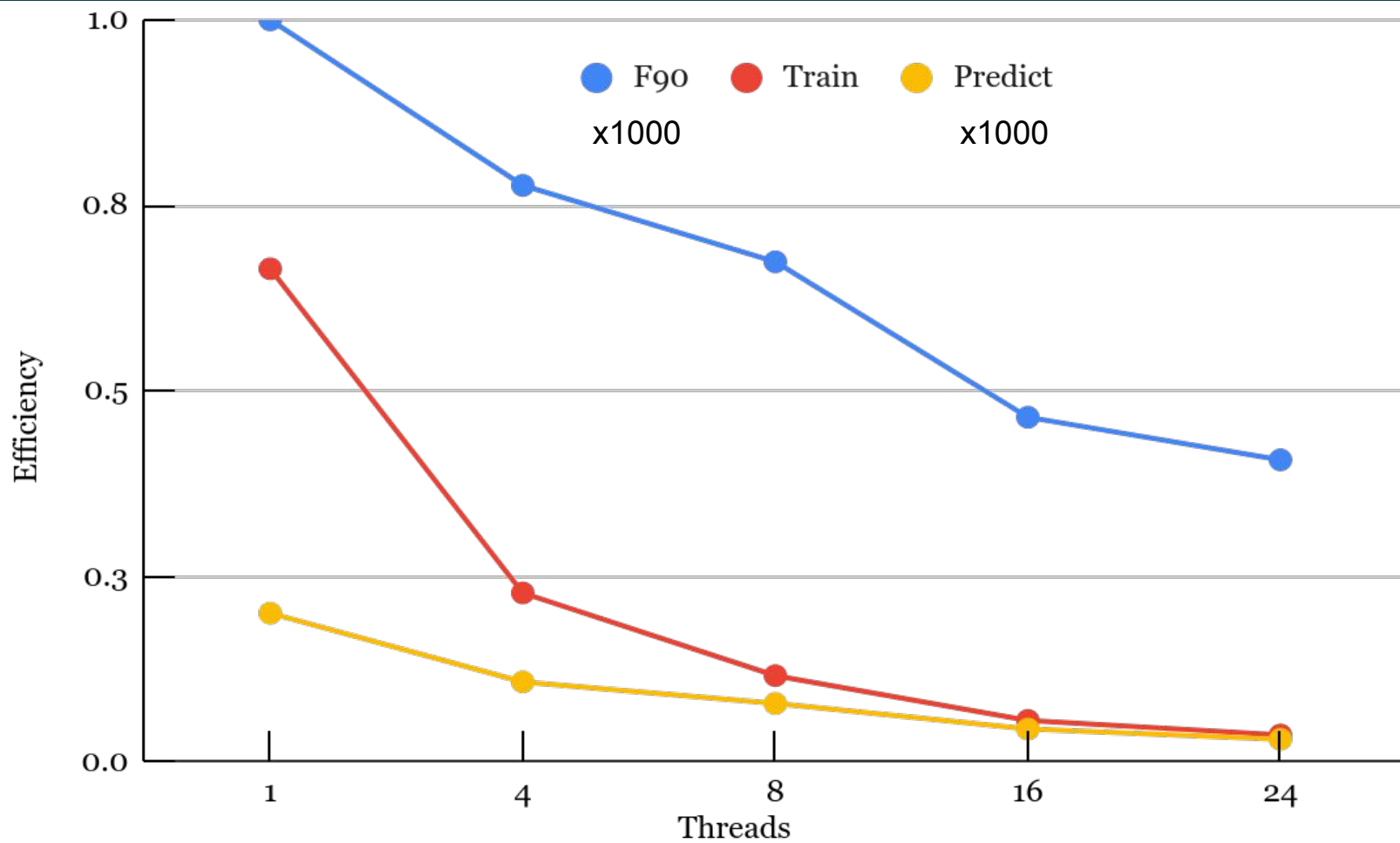
Tempos de processamento[s]



Speedup



Eficiência paralela



5

Considerações Finais

Considerações finais

- No trabalho foram usadas 2 abordagens, PINN e GQM
- O caso de teste foi executado no SDumont
- TensorFlow/Python e F90 com OpenMP
- A abordagem numérica teve melhor desempenho
- Trabalhos futuros
 - Avaliar o uso de outras arquiteturas ANN
 - Variar hiperparâmetros
 - Usar GPU



Obrigado!

Código fonte: <https://github.com/efurlanm/421/tree/main/project>