



Comparação de abordagens de computação de alto desempenho no ambiente Python para um estudo de caso de estêncil de cinco pontos

XV Brazilian e-Science Workshop (BreSci)

Autores: Eduardo F. Miranda, Stephan Stephany

Conteúdo

1. Introdução

2. Abordagens e recursos

3. Estudo de caso

4. Resultados



Introdução

Por que Python?

Prototipagem rápida

Interfaceamento fácil

Fácil de ler e manter

Comunidade código aberto

Muitas bibliotecas disponíveis (incluindo PAD)

Language Ranking: IEEE Spectrum

Rank	Language	Score
1	Python	100.0
2	Java	96.3
3	C	94.4
4	C++	87.5
5	JavaScript	79.4

<http://spectrum.ieee.org>

Módulo para Deep Learning / IA:

https://sdumont.incc.br/support_manual.php

```
module load deepl/deeplearn-py3.7
```

(TensorFlow 1.13, Keras, PyTorch)

Exemplo de pacote de desenvolvimento Python Intel



The screenshot shows the Intel Specialized Development Tools website. The header includes the Intel logo, the text "Specialized Development Tools", and navigation links for "USA (English)", "Sign In", and a search icon. Below the header, there is a breadcrumb trail "Home / Tools". The main content area features a large, glowing blue Python logo composed of multiple parallel lines. Below the logo, the text reads "Intel® Distribution for Python*", followed by a description: "Accelerate Python* and speed up core computational packages with this performance-oriented distribution. Powered by Anaconda* and available on conda*, PIP*, APT GET, YUM, and Docker*." At the bottom of the main content area, there is a blue button labeled "Choose & Download".

Specialized Development Tools

intel

USA (English) Sign In

Home / Tools

Intel® Distribution for Python*

Accelerate Python* and speed up core computational packages with this performance-oriented distribution. Powered by Anaconda* and available on conda*, PIP*, APT GET, YUM, and Docker*.

Choose & Download

Objetivos (I)

Python -> programação e prototipagem rápida, mas permite processamento de alto desempenho (PAD)?

Compilação do código Python padrão para linguagem intermediária (bytecode)

Execução interpretada (lenta) por Python

Então Python não é adequado PAD?

Objetivos (II)

Avaliam-se aqui abordagens de PAD para Python para estudo de caso específico objetivando:

Explorar diferentes abordagens PAD para Python utilizando a **biblioteca MPI**, executadas no supercomputador SDumont/LNCC

Avaliar o desempenho das abordagens

Referência: implementações F90 sequencial e paralela

Roteiro para o uso de recursos PAD em Python

2

Algumas abordagens e recursos Python para PAD

2.1 - IPython Parallel

IPython é um *shell* para computação interativa

IPython Parallel é um pacote com uma coleção de scripts para controlar clusters

Instâncias IPython em paralelo, interativamente

Suporta paralelismo **MPI**, threads (cores/GPU), etc.

Aplicativos paralelos desenvolvidos, executados, depurados e monitorados interativamente

2.2 - Biblioteca *MPI for Python*

Paralelização usando biblioteca de comunicação por troca de mensagens **MPI** (Message Passing Interface)

Permite execução em um um mais nós de memória compartilhada de supercomputador

Permite comunicação **MPI** de arrays NumPy

2.3 - Compilador Cython

Compila código-fonte Cython/Python para código C, que é então compilado por um compilador C para linguagem de máquina

Desempenho depende da portabilidade das chamadas Python originais para as chamadas C/C++ correspondentes

Permite interagir com outras bibliotecas otimizadas na linguagem C/C ++

2.4 - Biblioteca PyTorch

Biblioteca de código aberto para aprendizado de máquina (Facebook)

Suporta arrays/tensores multidimensionais

Permite a execução em CPU ou GPU

2.5 - Biblioteca NumPy

Desenvolvida principalmente para arrays multidimensionais

Útil para aplicações científicas

Possui funções para álgebra linear, transformada de Fourier, etc.

Possui interface para Fortran (F2PY)

2.6 - F2PY (integrada à biblioteca NumPy)

Permite criar bibliotecas Python (compostas de módulos) a partir de códigos F90

Requer poucas modificações no código F90

Módulos da biblioteca podem ser usados no código Python

Boa integração com o ambiente Python

2.7 - Biblioteca PyCuda

Acesso a API CUDA padrão usando Python

Porta partes intensivas do código Python para execução em GPU

Parte não-intensiva do código Python executado de forma interpretada

2.8 - Compilador Numba

Permite compilação AOT ou JIT

Suporta compilação de parte de Python e Numpy

Porta partes intensivas do código Python para código de máquina executável em processador multicore ou GPU

Parte não-intensiva do código Python executado de forma interpretada

2.9 - Implementação padrão de Python

Execução interpretada e portanto lenta

Útil para prototipagem rápida ou prova de conceito

Portabilidade quase plena para diferentes plataformas

Otimização de desempenho feita numa segunda etapa especificamente para a plataforma (exemplo: GPU)

2.10 - Código F90 de referência

Código F90 sem Python

Compilador GNU/gfortran (flag -O3)

Também compatível com Jupyter Notebook para desenvolvimento, compilação, execução, e análise dos resultados

Implementações: sequencial e paralela **MPI**

2.11 - Outras abordagens de PAD para Python

Incluem computação distribuída, computação em nuvem ou grade, ambiente específicos (Dask), etc.

Abordagens existentes ativamente desenvolvidas e várias outras sendo propostas...

2.12 - Aplicativo Jupyter Notebook

Aplicativo web de código aberto para computação interativa

Permite criar documento com texto descritivo, equações LaTeX, trechos de código executável, etc.

Arquitetura servidor-cliente com servidor executado em uma máquina remota ou local

Permite executar programas, upload/download de arquivos, interfacear com sistemas de submissão de jobs (SLURM, etc.) e outras funcionalidades

2.13 - Distribuição Anaconda

Suporta Python e R para computação científica e inclui o aplicativo Jupyter Notebook

Inclui aplicativos de ciência de dados e aprendizado de máquina, processamento de dados em grande escala, análise preditiva, etc.

Utilizado neste trabalho no ambiente do supercomputador SDumont

3

Estudo de caso

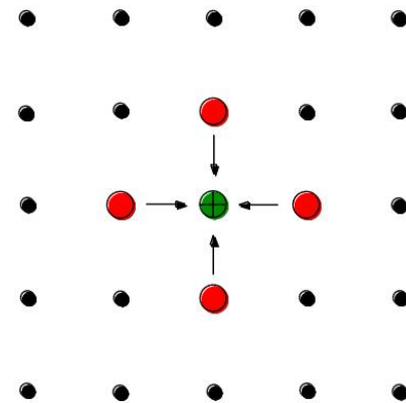
3.1 - Estudo de caso

Estudo de caso selecionado

Problema de difusão de calor 2D modelado com a equação de Poisson e resolvido pelo método das diferenças finitas usando um estêncil de 5 pontos. Simulação ao longo de um número de **timesteps** em que a grade 2D é sucessivamente atualizada

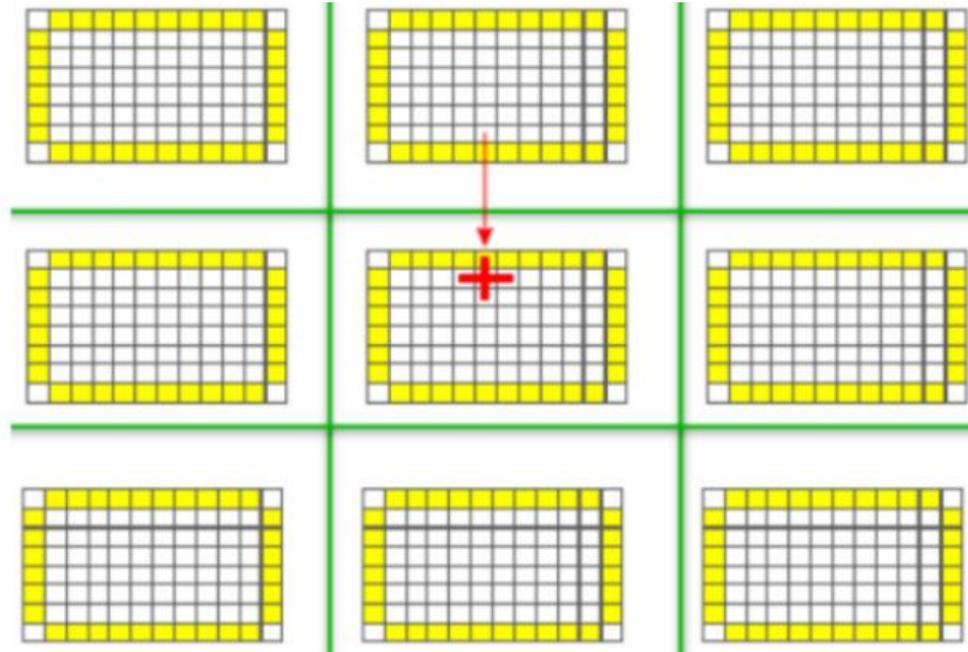
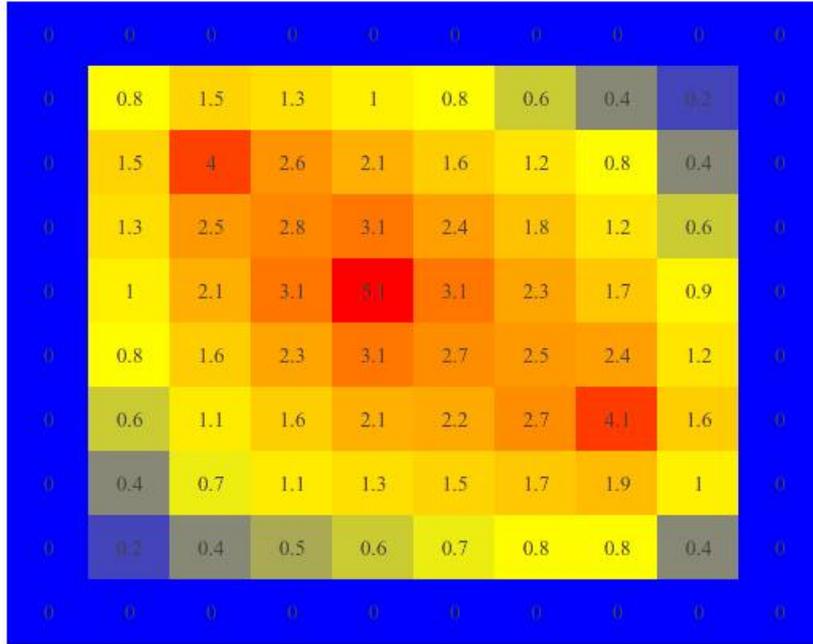
3.2 - Equação de Poisson 2D (difusão de calor)

Campo de temperatura U definido em malha discreta 2D (x, y) com resolução $\Delta x = \Delta y = h$ e estêncil de 5 pontos



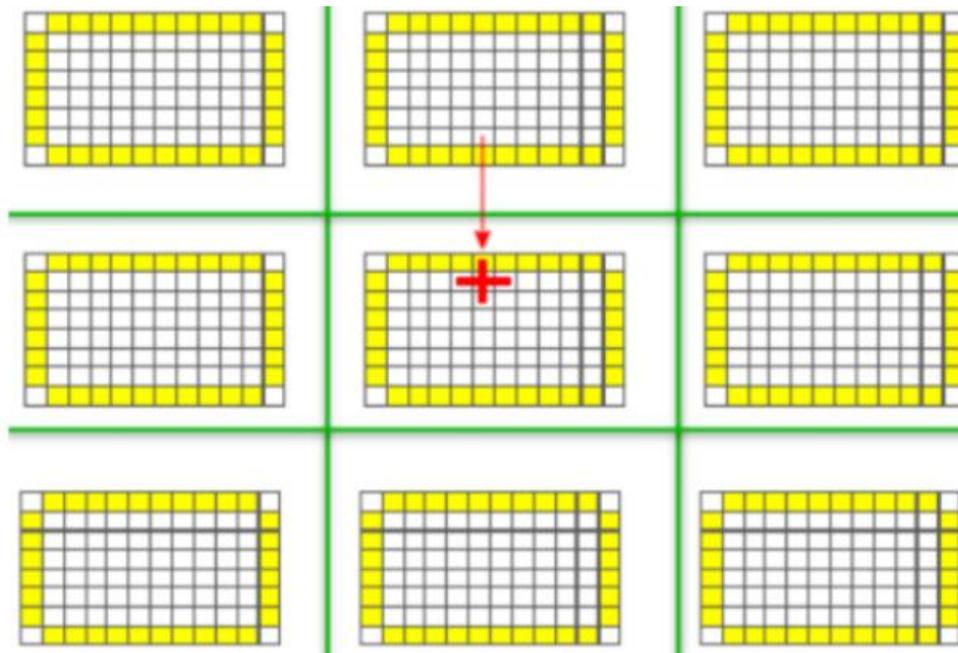
$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} \approx \frac{U_{i+1,j} + U_{i,j+1} - 4U_{i,j} + U_{i-1,j} + U_{i,j-1}}{h^2}$$

3.3 - Divisão do domínio 2D



Paralelização: domínio original é dividido em subdomínios com replicação das bordas nas fronteiras entre eles

3.4 - Atualização do domínio a cada timestep



A cada **timestep**, a atualização da grade com o estêncil de 5 pontos exige comunicação MPI das temperaturas nas bordas para subdomínios vizinhos

3.4 - Parte intensiva do código em questão

```
do j=2,by+1
  do i=2,bx+1
    anew(i,j)=1/2*(aold(i,j)+1/4*(aold(i-1,j)+aold(i+1,j)+aold(i,j-1)+aold(i,j+1)))
  enddo
enddo
```

F90

```
cpdef kernel(double[:,::1] anew, double[:,::1] aold, Py_ssize_t by, Py_ssize_t bx):
    for i in range(1,bx+1):
        for j in range(1,by+1):
            anew[i,j]=1/2*(aold[i,j]+1/4*(aold[i-1,j]+aold[i+1,j]+aold[i,j-1]+aold[i,j+1]))
```

Cython

```
@njit
def kernel(anew, aold):
    anew[1:-1,1:-1]=(
        1/2*(aold[1:-1,1:-1]+1/4*(aold[2:,1:-1]+aold[:-2,1:-1]+aold[1:-1,2:]+aold[1:-1,:-2])))
```

Numba

4

Resultados: Análise de desempenho sequencial e paralelo

4.1 - Ambiente do supercomputador SDumont

Nó B710: 2 procs. **Xeon E5-2695v2** 12-core

Nó B715: 2 procs. **Xeon E5-2695v2** 12-core + 2 **Tesla K40**

Nó Sequana X: 2 procs. **Xeon 6152** 22-core + 4 **Volta V100**

GNU Fortran 7.4, GNU Fortran 8.3, OpenMPI 4.0.1, Intel Fortran 19.0.3, Intel MPI, Python 3.6.12, Cython 0.29.20, NumPy 1.18.1, Numba 0.41.0, e CUDA 10.1 e outros

4.3 - Tempos de processamento (em segundos)

	Número de processos MPI								
	Seq.	1	4	9	16	36	49	64	81
F90	19.3	21.9	7.3	6.2	4.7	2.1	1.9	1.2	1.7
F2Py	18.9	23.6	7.5	6.2	4.6	2.1	1.6	1.3	1.0
Cython	24.0	24.0	7.5	6.3	4.7	2.2	1.7	1.3	2.1
Numba (CPU)	30.5	30.5	8.2	6.3	5.9	3.2	2.7	1.8	2.1
Python	212.4	227.2	64.7	44.8	33.5	15.2	10.4	7.8	6.7

4.4 - Speedup

Número de processos MPI

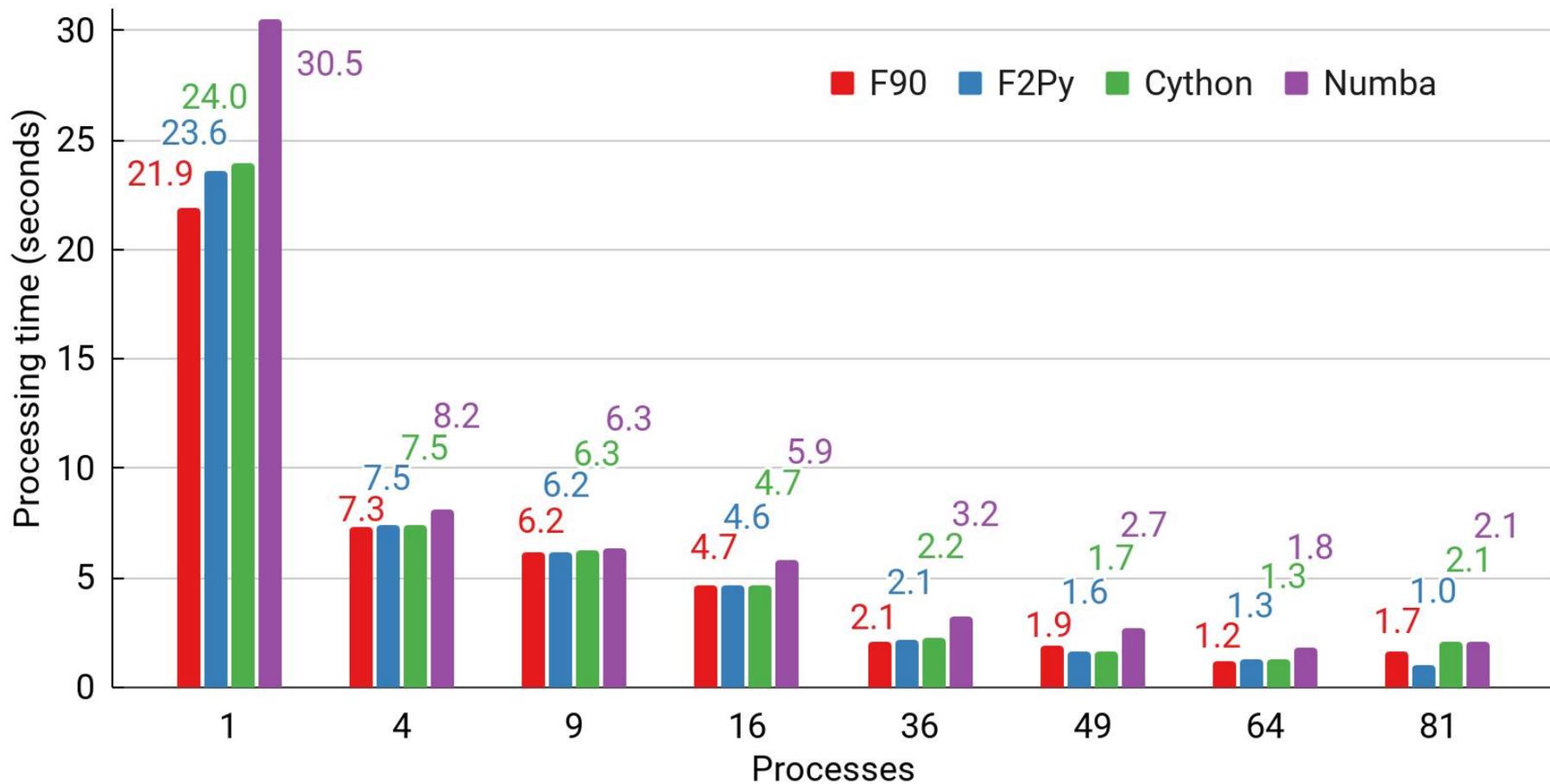
	Seq.	1	4	9	16	36	49	64	81
F90	1.0	0.9	2.6	3.1	4.1	9.0	10.2	15.7	11.4
F2Py	1.0	0.8	2.6	3.1	4.2	9.0	11.8	15.1	19.0
Cython	0.8	0.8	2.6	3.1	4.1	8.6	11.6	14.7	9.4
Numba (CPU)	0.6	0.6	2.4	3.0	3.3	6.0	7.2	10.8	9.3
Python	0.1	0.1	0.3	0.4	0.6	1.3	1.8	2.5	2.9

4.5 - Eficiência paralela

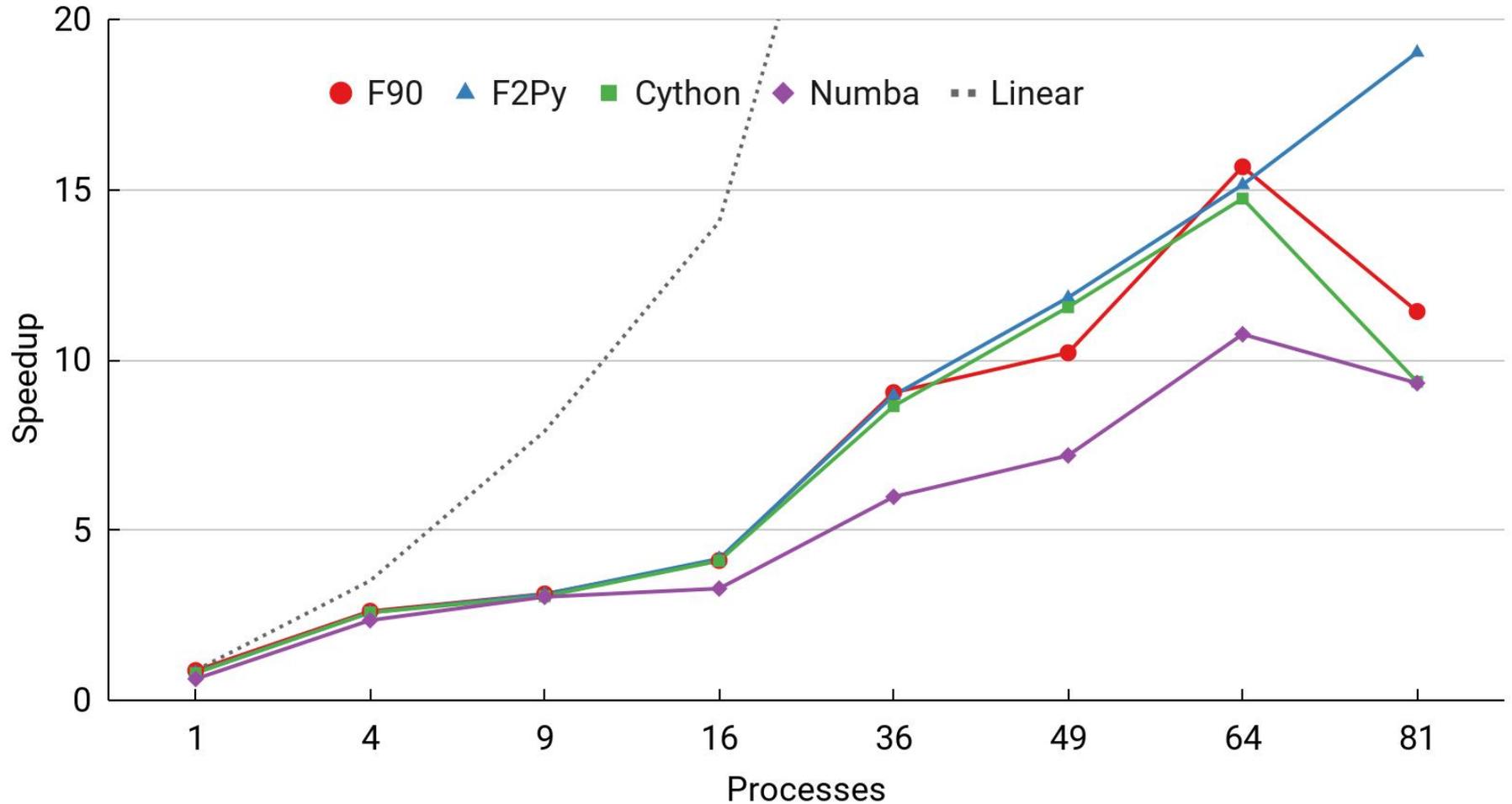
Número de processos MPI

	Seq.	1	4	9	16	36	49	64	81
F90	1.00	0.88	0.66	0.35	0.26	0.25	0.21	0.24	0.14
F2Py	1.02	0.82	0.65	0.35	0.26	0.25	0.24	0.24	0.23
Cython	0.80	0.80	0.65	0.34	0.26	0.24	0.24	0.23	0.12
Numba (CPU)	0.63	0.63	0.59	0.34	0.21	0.17	0.15	0.17	0.12
Python	0.09	0.08	0.07	0.05	0.04	0.04	0.04	0.04	0.04

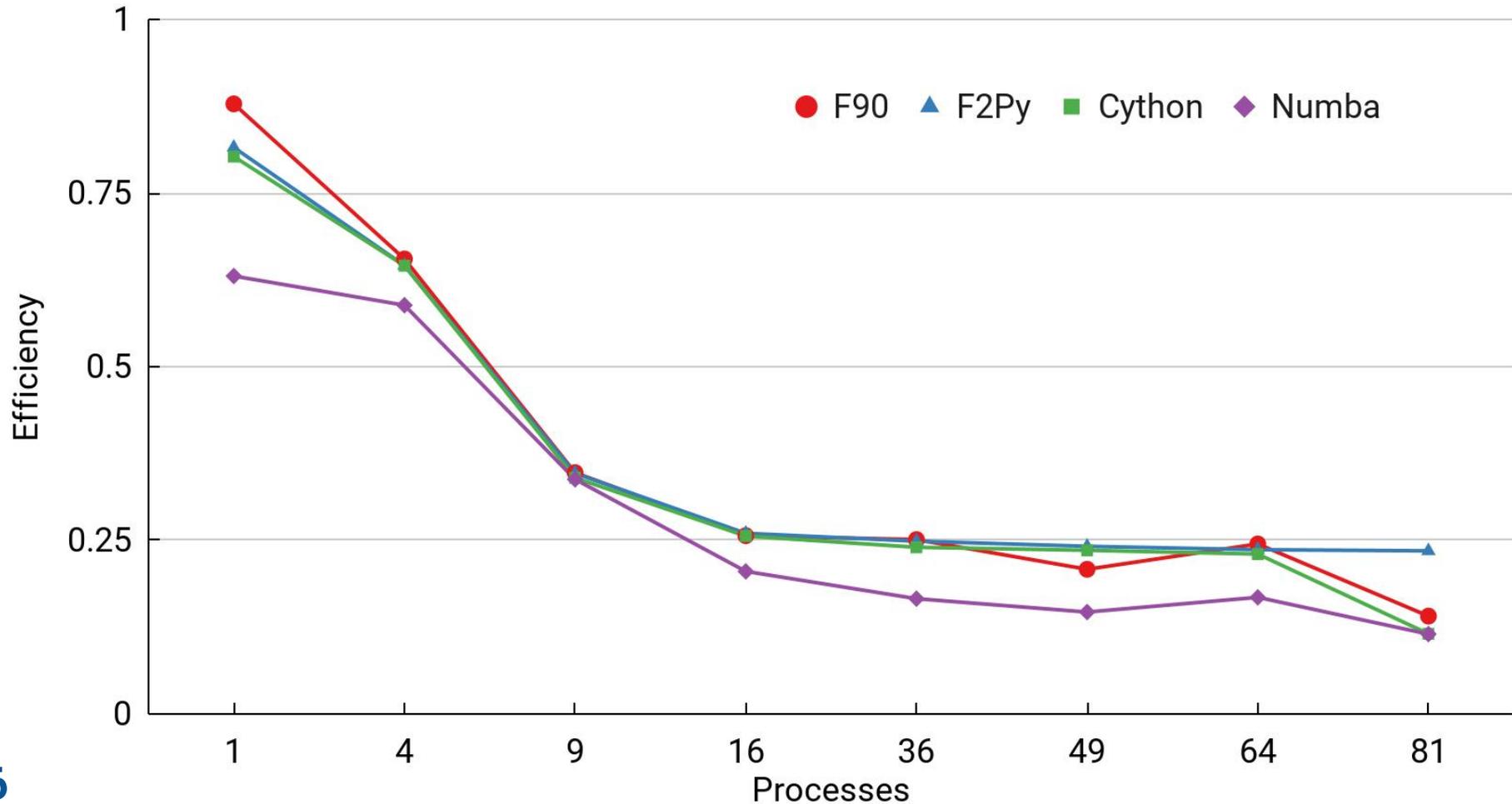
4.6 - Tempos de processamento (em segundos)



4.7 - Speedup



4.8 - Eficiência paralela



4.9 - Comparação tempos (s) Numba-GPU x F90

	GPU	Seq.	1	4	9	16
F90/B715		<u>19.3</u>	21.9	<u>7.3</u>	6.2	4.7
F90/Seq-X		15.8	15.6	4.1	2.1	1.5
Numba-GPU/B715	<u>22.3</u>					
Numba-GPU/Seq-X	<u>8.0</u>					

Agradecimentos

Ao LNCC (Laboratório Nacional de Computação Científica), projeto 205341 AMPEMI (2020-I), para uso do supercomputador Santos Dumont (nó do SINAPAD, o Sistema Nacional de PAD)



Obrigado!

Código fonte: <https://github.com/efurlanm/bs21>

Contato: Eduardo Furlan Miranda. Programa de Pós Graduação em Computação Aplicada (CAP) / INPE

E-mail: eduardo.miranda@inpe.br

Contato: Dr. Stephan Stephany. Coordenação de Pesquisa Aplicada e Desenvolvimento Tecnológico (COPDT) (CAP) / INPE

E-mail: stephan.stephany@inpe.br

DOI: [10.5281/zenodo.10672456](https://doi.org/10.5281/zenodo.10672456)