Avaliação de Frameworks de HPC em Python: Uma Análise Comparativa entre NumPy, CuPy, PyTorch e Numba Usando Operações Numéricas Vetorizadas

Marcos Eduardo Lopes Silva, Claudio Schepke

[1]Engenharia de Software – Universidade Federal do Pampa (UNIPAMPA)
Alegrete – RS – Brasil

{marcoseduardo.aluno,claudioschepke}@unipampa.edu.br

Resumo. *A computação de alto desempenho (HPC) faz uso crescente de aceleradores como GPUs. Entretanto, programar diretamente em CUDA é uma tarefa trabalhosa e complexa. Python, por sua simplicidade e grande ecossistema, oferece diversas ferramentas capazes de explorar GPUs de forma transparente. Este trabalho realiza uma avaliação comparativa entre NumPy, CuPy, PyTorch e Numba, aplicando um mesmo conjunto de operações numéricas vetorizadas. Os resultados indicam que as soluções baseadas em GPU apresentam ganhos expressivos para tamanhos maiores de entrada.*

1. Introdução

Embora linguagens de baixo nível como C++ e Fortran sejam tradicionalmente usadas para tarefas computacionalmente intensivas devido à sua eficiência, a facilidade de uso e a extensa biblioteca padrão do Python o tornaram uma escolha popular para computação científica, apesar de sua natureza interpretada [Raschka et al. 2020].

Nos últimos anos, diversos frameworks têm surgido com o objetivo de permitir que Python utilize recursos de GPU sem a necessidade de escrever kernels CUDA manualmente. Entre eles, destacam-se NumPy, CuPy, PyTorch e Numba, cada um possuindo estratégias distintas para acelerar operações numéricas. NumPy representa a abordagem tradicional, executando operações vetorizadas em CPU e servindo como referência para comparação. CuPy oferece uma interface quase idêntica à do NumPy, porém executa suas operações diretamente em GPUs NVIDIA [Nishino and Loomis 2017]. PyTorch, originalmente projetado para aprendizado de máquina, inclui implementações altamente otimizadas para GPU e CPU [Paszke et al. 2019]. Numba, por sua vez, utiliza compilação JIT baseada em LLVM e permite a criação de kernels customizados com suporte tanto para CPU quanto GPU [Lam et al. 2015].

O objetivo deste trabalho é comparar o desempenho dessas quatro tecnologias ao executar um mesmo algoritmo numérico, analisando tempo de execução e ganho de desempenho conforme o tamanho do problema aumenta.

2. Metodologia

Os experimentos foram realizados em um ambiente Google Colab equipado com uma GPU NVIDIA Tesla T4 com 16 GB de memória. O algoritmo selecionado consiste em operações vetorizadas simples, que aparecem frequentemente em aplicações científicas e de engenharia. Para cada valor N, gerou-se um vetor de entrada contendo valores

igualmente espaçados, sobre o qual foram aplicadas operações de multiplicação, soma e exponenciação, seguidas de uma redução final por soma.

A lógica central do algoritmo, independentemente do framework utilizado, segue a expressão matemática $A = x^2 + 2x$, seguida do cálculo de $B = \exp(A)$ e, finalmente, da soma total dos elementos resultantes. Esse procedimento foi implementado separadamente em NumPy, CuPy, PyTorch e Numba, respeitando as particularidades de cada tecnologia. No caso dos frameworks que utilizam GPU, foram incluídos implicitamente os custos de transferência de dados quando necessários, permitindo uma análise mais realista do comportamento de cada solução.

Os tempos de execução foram medidos repetidamente para diferentes tamanhos de entrada, abrangendo desde vetores relativamente pequenos até entradas significativamente maiores. Todos os resultados foram armazenados em arquivos CSV. Posteriormente, foram calculados os valores de *speedup*, tomando NumPy como referência. A partir desses dados, foram gerados gráficos de tempo de execução e de ganho de desempenho, ambos fundamentais para avaliar a escalabilidade e a eficiência relativa dos frameworks.

3. Resultados

Os resultados demonstram diferenças claras entre os frameworks, especialmente à medida que o tamanho do problema aumenta. As execuções em CPU, representadas pelo NumPy, apresentaram tempos menores quando o tamanho dos vetores era reduzido, devido ao baixo custo inicial e à ausência de transferência de dados. No entanto, à medida que o valor de N cresceu, o tempo de execução aumentou significativamente.

A Figura 1 exibe o comportamento do tempo de execução em escala log-log. Observa-se que as abordagens baseadas em GPU apresentam maior latência inicial, mas superam o desempenho do NumPy quando o volume de dados cresce. CuPy e PyTorch demonstraram desempenho bastante similar, com CuPy apresentando ligeira vantagem nos maiores tamanhos.

A Figura 2 apresenta os valores de speedup em relação ao NumPy. O comportamento observado confirma o padrão identificado nos tempos de execução: para tamanhos reduzidos, o uso de GPU geralmente não compensa o custo inicial, mas, a partir de certo ponto, o ganho de desempenho se torna significativo. CuPy obteve os maiores speedups, seguido de perto por PyTorch. Numba apresentou comportamento mais variável, refletindo sua dependência de implementações específicas de kernels.

4. Discussão

Os resultados permitem algumas interpretações relevantes para a escolha prática de frameworks em aplicações de HPC com Python. Para tarefas pequenas, a execução em CPU continua mais eficiente, uma vez que o tempo gasto para configurar o ambiente de GPU e transferir dados tende a ser superior ao tempo de cálculo propriamente dito. À medida que o tamanho do problema cresce, frameworks como CuPy e PyTorch demonstram grande vantagem, alcançando aceleração significativa sem exigir modificações profundas no código. CuPy se destaca particularmente pela similaridade com a API do NumPy, permitindo migrações rápidas de código já existente. PyTorch, apesar de ser originalmente voltado ao aprendizado de máquina, mostrou desempenho competitivo e facilidade de uso.

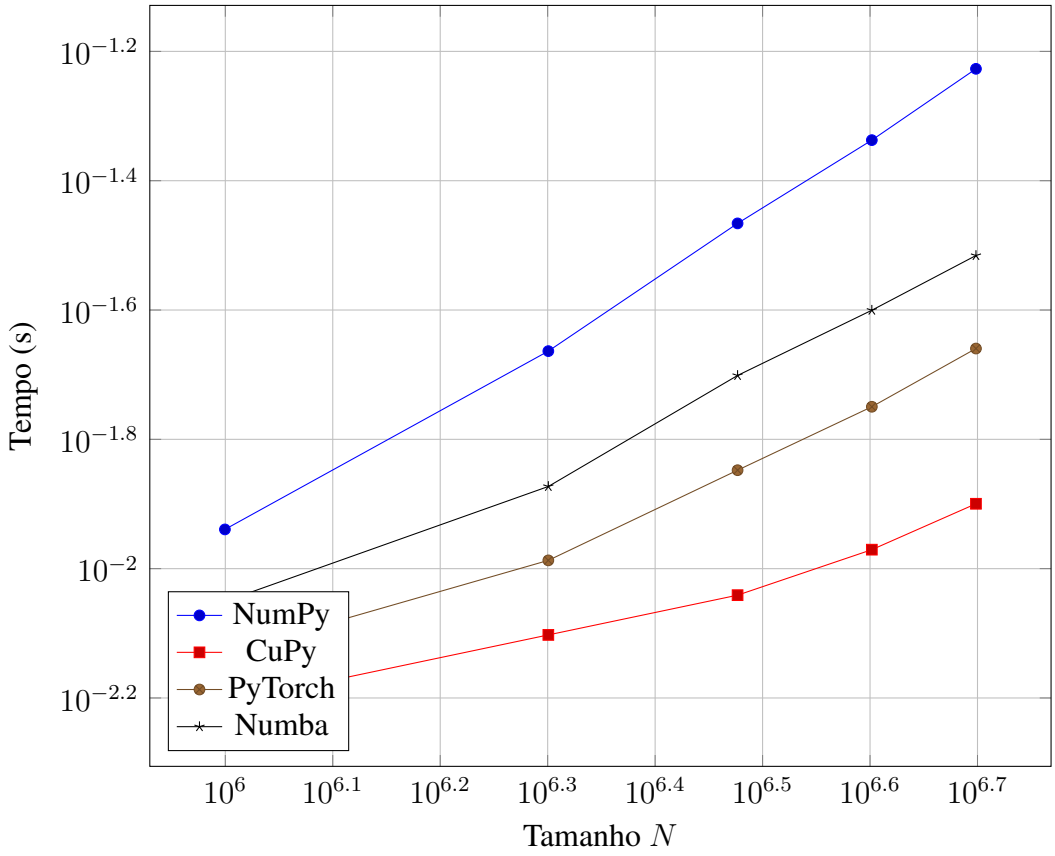

Figura 1. Tempo de execução em escala log-log.

5. Conclusão

Este trabalho investigou o desempenho de quatro frameworks populares utilizados para computação científica em Python. A avaliação experimental demonstrou que a utilização de GPU pode trazer ganhos significativos de desempenho, especialmente para tamanhos de entrada elevados. CuPy e PyTorch apresentaram os melhores resultados de forma consistente, enquanto Numba se mostrou mais dependente da configuração do kernel. NumPy, apesar de não explorar GPUs, continua eficiente para problemas de pequena escala e serve como referência sólida para comparações.

Os resultados reforçam a ideia de que Python, mesmo sendo uma linguagem de alto nível, pode servir como base para aplicações de HPC quando combinado com ferramentas apropriadas. A possibilidade de aproveitar GPUs sem escrever código CUDA manualmente abre espaço para aplicações mais acessíveis e produtivas, beneficiando tanto iniciantes quanto usuários experientes.

Referências

Lam, S. K., Pitrou, A., and Seibert, S. (2015). Numba: a LLVM-based Python JIT compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, New York, NY, USA. Association for Computing Machinery.

Nishino, R. and Loomis, S. H. C. (2017). Cupy: A numpy-compatible library for nvidia gpu calculations. *31st confernce on neural information processing systems*, 151(7).

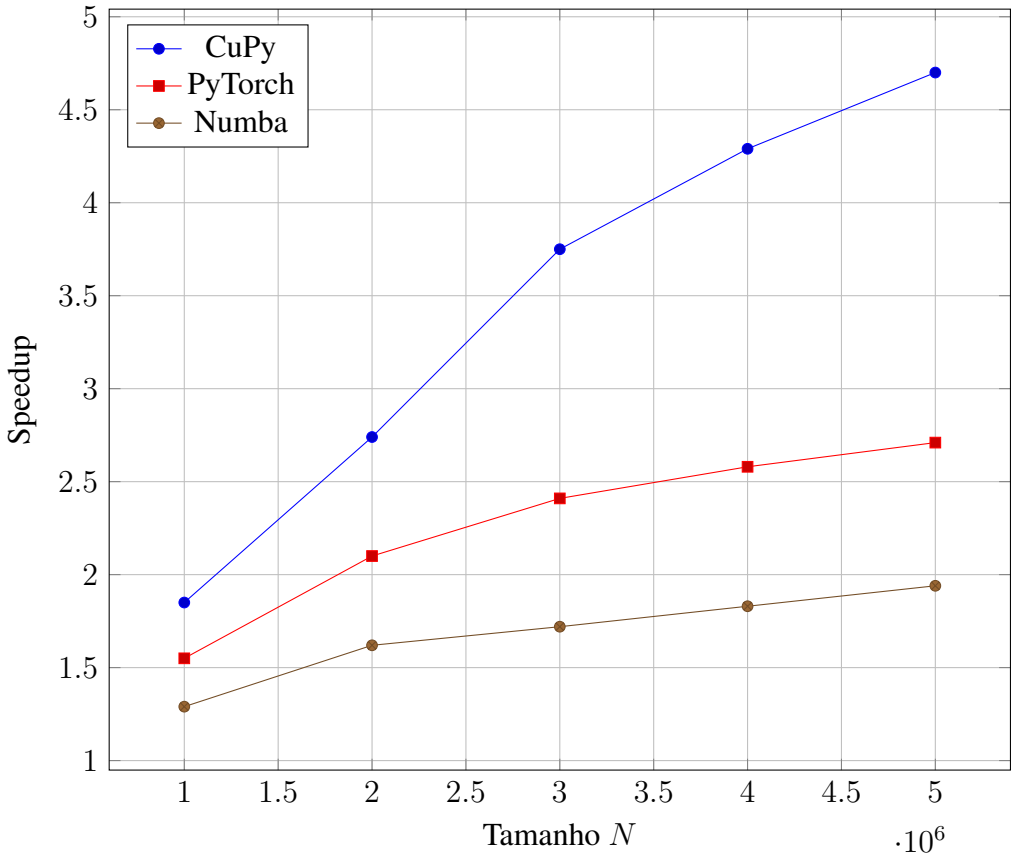

Figura 2. Speedup relativo ao NumPy.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Raschka, S., Patterson, J., and Nolet, C. (2020). Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence. *Information*, 11(4).