UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMATICA

ALIELSON FERREIRA PEQUENO

AVALIAÇÃO DO USO DE PARALELISMO EM PYTHON PARA UM PREDITOR DE INTERAÇÕES ATÔMICAS DE ESTRUTURAS TRIDIMENSIONAIS DE PROTEÍNAS

JOÃO PESSOA
2022

ALIELSON FERREIRA PEQUENO

AVALIAÇÃO DO USO DE PARALELISMO EM PYTHON PARA UM PREDITOR DE INTERAÇÕES ATÔMICAS DE ESTRUTURAS TRIDIMENSIONAIS DE PROTEÍNAS

Monografia apresentada ao curso Ciência da Computação do Centro de Informática, da Universidade Federal da Paraíba, como requisito para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Thais Gaudêncio do Rego

JOÃO PESSOA

2022

Catalogação na publicação
Seção de Catalogação e Classificação

P425a Pequeno, Alielson Ferreira.
Avaliação do uso de paralelismo em Python para um preditor de interações atômicas de estruturas tridimensionais de proteínas / Alielson Ferreira Pequeno. - João Pessoa, 2023.
35 f. : il.

Orientação: Thais Gaudencio Rêgo.
TCC (Graduação) - UFPB/CI.

1. Bioinformática estrutural. 2. Proteômica. 3. Multiprocessamento. 4. Python. I. Rêgo, Thais Gaudêncio. II. Título.

UFPB/CI CDU 004.43

Elaborado por Michelle de Kássia Fonseca Barbosa - CRB-738

ALIELSON FERREIRA PEQUENO

AVALIAÇÃO DO USO DE PARALELISMO EM PYTHON PARA UM PREDITOR DE INTERAÇÕES ATÔMICAS DE ESTRUTURAS TRIDIMENSIONAIS DE PROTEÍNAS

Monografia apresentada ao curso Ciência da Computação do Centro de Informática, da Universidade Federal da Paraíba, como requisito para a obtenção do grau de Bacharel em Ciência da Computação.

João Pessoa, 29 de Março de 2022

BANCA EXAMINADORA

_____

Prof. Dra. Daniela Coelho Batista Guedes Pereira

Universidade Federal da Paraíba - UFPB

_____

Prof. Dra. Thaís Gaudencio do Rêgo

Universidade Federal da Paraíba - UFPB

_____

Prof. Dr. Marcelo Iury De Sousa Oliveira

Universidade Federal da Paraíba - UFPB

Dedico este trabalho aos meus pais e amigos que sempre me incentivaram.

“Se um homem não sabe para qual porto navega, nenhum vento lhe é favorável.” – Sêneca

RESUMO

Com os avanços tecnológicos alcançados na área da computação, o paralelismo vem consequentemente sendo cada vez mais utilizado. Nesse contexto, ferramentas que calculam distância e interações atômicas, normalmente, são codificadas sem o uso desse artifício. Dessa forma, o objetivo deste trabalho é otimizar uma dessas ferramentas, o Ysera, para avaliação do impacto do uso do paralelismo em seu código. A otimização foi realizada com a reestruturação do seu código para que trabalhasse de forma paralela com recursos disponíveis para a linguagem Python, o multiprocessamento. A avaliação de ganho de desempenho foi feita através de diversas execuções do software de forma sequencial e de forma paralela, utilizando arquivos de entrada no formato PDB de tamanhos variados. Avaliando seu tempo de execução, consumo de memória e processamento. Também foi avaliado o impacto que o aumento do número de processos causa em seu desempenho. Como resultado, obteve-se uma melhora de desempenho de até mais de 50% comparando com a versão original da ferramenta, reduzindo o tempo computacional para obter os dados de saída desejados.

Palavras-chave: Bioinformática Estrutural, Proteômica, Multiprocessamento, Python

ABSTRACT

With the technological advances achieved in the field of computing, parallelism has consequently been increasingly used. In this context, tools that calculate distance and atomic interactions are usually coded without using this device. Thus, this work aims to optimize one of these tools, Ysera, to evaluate the impact of using parallelism in your code. The optimization was carried out with the restructuring of its code so that it worked in parallel with resources available for the Python language, the multiprocessing. The performance gain evaluation was done through several software executions sequential and parallel, using input files in PDB format of different sizes. Evaluating its execution time, memory consumption, and processing. The impact that the increase in the number of processes causes on its performance was also evaluated. As a result, a performance improvement of up to more than 50% was obtained compared to the original version of the tool, reducing the computational time to obtain the desired output data.

Keywords: Bioinformatics Structural, Proteomics,Multiprocessing,Python

LISTA DE ILUSTRAÇÕES

Figura 1 — Processso na memória 15
Figura 2 — Exemplo de concorrência 16
Figura 3 — Exemplo de paralelismo 17
Figura 4 — Exemplo de multiprocessamento 18
Figura 5 — Estrutura de repetição dupla, imagem tirada da versão original do Ysera. 22
Figura 6 — Função estruturada através da estrutura de repetição tirada da versão original do Ysera. 23
Figura 7 — Funções sendo chamadas para serem executadas de forma paralela em uma pool de processos. 24
Figura 8 — Estruturas tridimensionais dos PDBs utilizados neste trabalho geradas pelo PISA (software) 25
Figura 9 — Tempos de execução com o numero de processos por pool. 30
Figura 10 — Gráfico Linear obtido com as implementações paralelas. Onde a linha linear significa a otimização perfeita e o multiprocessing a otimização alcançada neste trabalho. .. 31

LISTA DE TABELAS

Tabela 1 — Resumo dos trabalhos relacionados 20
Tabela 2 — Dados das execuções em série 28
Tabela 3 — Dados das execuções paralelas 28

LISTA DE ABREVIATURAS E SIGLAS

PDB	Protein Data Bank

SUMÁRIO

1	**INTRODUÇÃO**	12
1.1	**CONTEXTUALIZAÇÃO E MOTIVAÇÕES**	12
1.2	OBJETIVO GERAL	13
1.3	ESTRUTURA DA MONOGRAFIA	13
2	**FUNDAMENTAÇÃO TEÓRICA**	14
2.1	PROCESSO	14
2.2	THREADS	15
2.3	CONCORRÊNCIA E PARALELISMO	15
2.4	MULTIPROCESSAMENTO	17
2.5	CONSEQUÊNCIAS DO MULTIPROCESSAMENTO	18
2.6	YSERA	18
2.7	TRABALHOS RELACIONADOS	19
3	**METODOLOGIA**	22
3.1	MATERIAIS	24
3.2	ARQUIVOS DE TESTE	25
3.3	AVALIAÇÃO	26
4	**APRESENTAÇÃO E ANÁLISE DOS RESULTADOS**	28
5	**CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS**	32
5.1	TRABALHOS FUTUROS	32
	REFERÊNCIAS	33

1 INTRODUÇÃO

A dificuldade no desenvolvimento de programas que funcionassem de forma paralela, e o alto custo e reduzida oferta de máquinas que utilizavam paralelismo em sua arquitetura, fizeram com que a programação paralela fosse um artifício pouco utilizado no passado. Contudo, vários fatores influenciaram uma mudança neste quadro no começo dos anos 2000, entre eles está a produção massiva de processadores mais baratos e rápidos, a evolução nas redes de computadores e a criação da computação distribuída [FARIA, 2023].

Uma maior capacidade de armazenamento, assim como a demanda por mais poder de processamentos, são outros fatores que influenciaram bastante no uso do paralelismo. Independente do quão avançado tecnologicamente um processador individual possa ser, ao reunir vários do mesmo para trabalharem em uma mesma tarefa em conjunto, o resultado será mais rápido. Quando se busca utilizar o paralelismo para se resolver algum problema, busca-se mais eficiência, logo linguagens como C, C# e Ada normalmente são escolhidas, mas não são as únicas opções.

Nos últimos anos a linguagem Python vem ganhando muito espaço na computação científica. É uma linguagem multi-paradigma, interpretada, que oferece uma grande variedade de bibliotecas científicas, embora não seja referência quando o assunto é alto desempenho. Ainda sim, a linguagem Python oferece pacotes de multiprocessamento que serão utilizados neste trabalho para otimizar uma ferramenta de bioinformática que apresenta uma massiva repetição de operações onde o paralelismo pode ser muito bem aproveitado para otimizá-la.

1.1 CONTEXTUALIZAÇÃO E MOTIVAÇÕES

O maior desafio encontrado ao paralelizar um código é a reestruturação desse algoritmo sequencial para um algoritmo paralelo, uma vez que, em muitas vezes, essa reestruturação não é possível, tornando-se necessário recriar todo o programa do zero [FARIA, 2023]. Na Bioinformática, ferramentas de cálculo de distâncias e interações atômicas, em sua maioria, são desenvolvidas de forma sequencial, consumindo maior processamento e tempo de execução.

Dessa forma, o Ysera, ferramenta de bioinformática que apresenta o mesmo objetivo e utilizado neste trabalho, teve seu código reestruturado para o uso de paralelismo e como

vantagem conseguiu-se reduzir o tempo computacional necessário para conseguir os dados que o software oferece. A ferramenta Ysera, que foi otimizada neste trabalho, assim como as demais que calculam as distâncias e interações atômicas possibilitam avanços biotecnológicos principalmente porque entender sobre a estruturas morfológica e características físicas da proteína é essencial na engenharia de ligações e de proteínas, impactando nas áreas de diagnóstico molecular, engenharia genética, engenharia metabólica, bioprocessos, desenvolvimento racional de fármacos, agropecuária e na indústria alimentícia.

1.2 OBJETIVO GERAL

O objetivo deste trabalho é utilizar técnicas de paralelismo e multiprocessamento disponíveis na linguagem Python em uma ferramenta de bioinformática para o cálculo de distâncias e interações atômicas, o Ysera, avaliando o impacto de seu uso.

1.3 ESTRUTURA DA MONOGRAFIA

No Capítulo 2 são abordados os temas fundamentais para a compreensão deste trabalho. Conceitos básicos como processo, thread, concorrência, paralelismo e multiprocessamento. O software Ysera, que será a referência para a otimização neste trabalho, é explicado. Por fim, são apresentados trabalhos relacionados à área.

No Capítulo 3 é apresentada a metodologia, explicando todos os passos que foram necessários para chegar nos resultados apresentados. Entre os passos descritos está a descrição das alterações feitas no software Ysera, a biblioteca utilizada, os arquivos de teste e as métricas utilizadas.

No Capítulo 4 são apresentados os resultados obtidos, a partir da execução dos passos descritos na metodologia. Serão exibidas e discutidas tabelas e gráficos com os resultados do multiprocessamento.

Por fim, no Capítulo 5, são apresentadas conclusões acerca do trabalho, considerando os pontos altos e baixos. Além disso, o tópico é finalizado ao comentar sobre trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

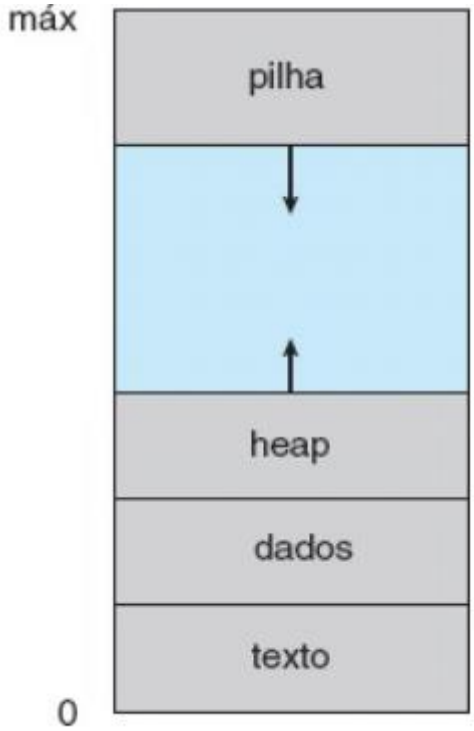
Nesta seção serão abordados os conceitos importantes para o entendimento e desenvolvimento deste trabalho. Conceitos básicos e essenciais de sistemas operacionais como processos, threads, concorrência e paralelismo. Também será abordado o que é multiprocessamento e no final será apresentando trabalhos relacionados que utilizam das ferramentas de paralelismo do Python para diminuir o tempo computacional.

2.1 PROCESSO

Com o contínuo avanço tecnológico na área da computação, hoje é padrão todo tipo de sistema operacional possuir a capacidade de carregar vários programas em memória e executá-los de forma concorrente, muito diferente do que acontecia nas primeiras gerações de computadores, onde os sistemas eram limitados, de forma que apenas um programa fosse executado de cada vez. Eles eram chamados de sistemas monotarefa, sendo o MS-DOS um exemplo desse tipo de sistema. Para alcançar essa evolução foi necessário um controle mais firme e uma compartimentalização dos diversos programas. Toda essa necessidade culminou na noção que temos hoje de processo, que de maneira informal é um programa em execução. Um processo é a unidade de trabalho em um sistema moderno de tempo compartilhado [SILBERSCHATZ, 2015].

Um processo é mais do que um programa em execução, ele também possui a atividade corrente, representada pelo valor do contador do programa e o conteúdo dos registradores do processador. Normalmente, um processo também agrega a pilha do processo que contém dados temporários, e uma seção de dados, que contém variáveis globais. Um processo também pode incluir um heap, que é a memória dinamicamente alocada durante o tempo de execução do processo [SILBERSCHATZ, 2015]. Na Figura 1 é apresentada a estrutura de um processo na memória. Na próxima seção veremos o que são threads e sua relação com os processos.

Figura 1 — Processso na memória



Fonte: Silberschatz, Galvin e Gagne (2015)

2.2 THREADS

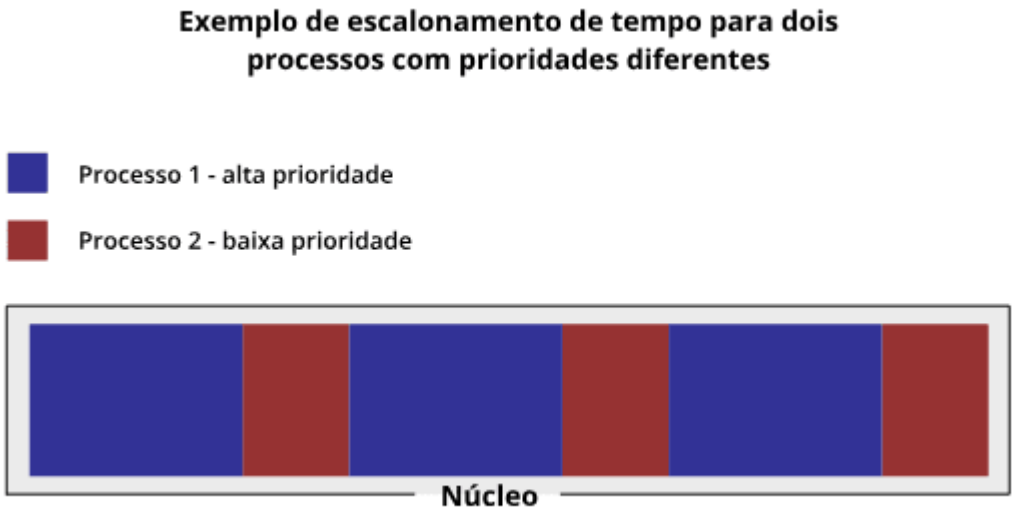
Dentro de cada processo existe uma linha de execução chamada Thread. Cada thread tem o seu próprio estado de processador e também tem sua própria pilha, mas acaba compartilhando a memória atribuída ao processo com as outras threads filhas do mesmo processo [TANENBAUM, 2016]. O uso de múltiplas threads acarreta em concorrência ou paralelismo, que detalharemos na próxima seção

2.3 CONCORRÊNCIA E PARALELISMO

Concorrência e Paralelismo são termos que costumam ser confundidos, no entanto, o primeiro trata sobre lidar com muitas coisas ao mesmo tempo. O segundo, paralelismo, é sobre fazer muitas coisas ao mesmo tempo [PIKE, 2023].

Concorrência é uma propriedade de software onde duas ou mais tarefas estão em progresso simultaneamente, elas progridem em um mesmo intervalo de tempo, como pode ser visto na Figura 2 [NAVARRO, 2014].

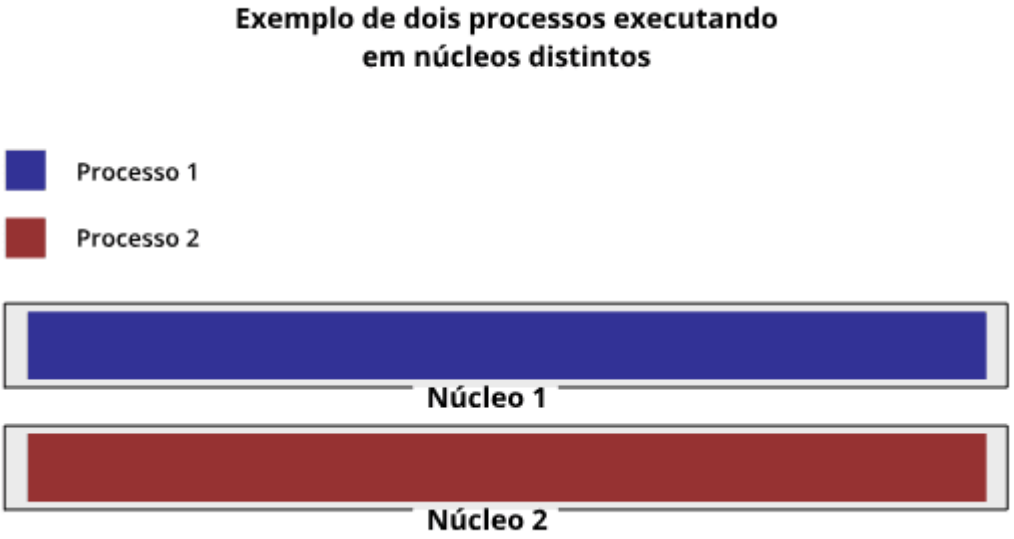
Figura 2 — Exemplo de concorrência



Fonte: https://dev.to/filipebeck/assincronismo-concorrencia-e-paralelismo-19hk#:~:text=Concorr%C3%AAncia%3A%20Refere%2Dse%20%C3%A0%20v%C3%A1rias,concorrendo%20por%20clocks%20da%20CPU.

Paralelismo é uma propriedade de tempo de execução onde duas ou mais tarefas estão sendo executadas simultaneamente, ou seja, paralelismo acaba implicando em concorrência, mas concorrência não implica em paralelismo.[NAVARRO, 2014] A Figura 3 exemplifica a execução simultânea de tarefas. Na próxima seção veremos como os processadores executam tarefas simultaneamente

Figura 3 — Exemplo de paralelismo

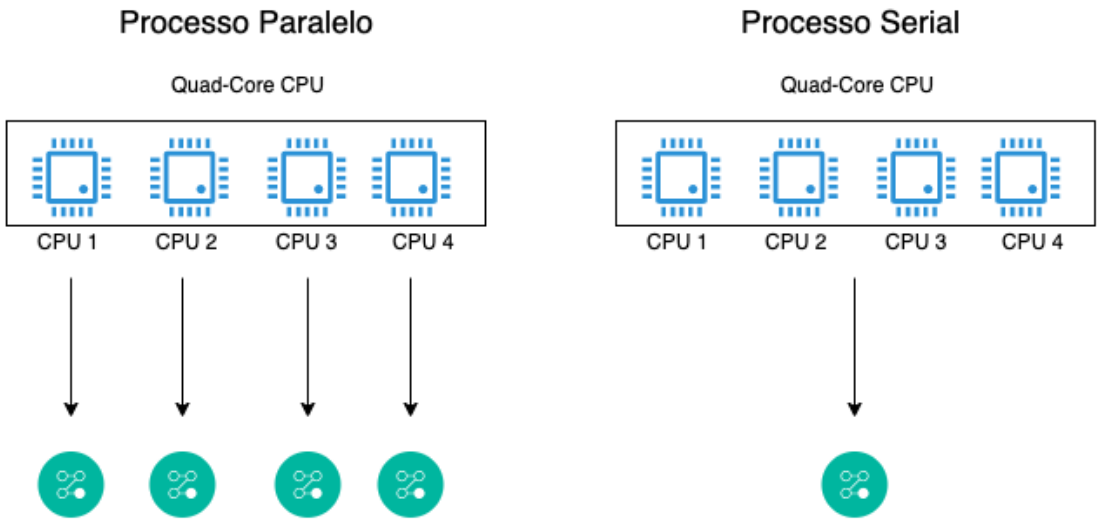


Fonte: https://dev.to/filipebeck/assincronismo-concorrencia-e-paralelismo-19hk#:~:text=Concorr%C3%AAncia%3A%20Refere%2Dse%20%C3%A0%20v%C3%A1rias,concorrendo%20por%20clocks%20da%20CPU.

2.4 MULTIPROCESSAMENTO

Multiprocessamento se refere a capacidade que alguns processadores possuem de realizar a execução paralela de dois ou mais processos. Multiprocessador se refere a um sistema de computadores que possui duas ou mais CPUs que compartilham o acesso total a uma memória RAM comum e também fazem a execução simultânea de processos [TANENBAUM, 2016]. A Figura 4 exemplifica processos sendo executados utilizando multiprocessamento e de forma serial. Na próxima seção abordaremos algumas das vantagens e desvantagens do uso do multiprocessamento.

Figura 4 — Exemplo de multiprocessamento



Fonte: https://medium.com/@adson.develop/uma-pequena-introdu%C3%A7%C3%A3o-a-programa%C3%A7%C3%A3o-paralela-e-multiprocessamento-com-python-232bbf72a8f7

2.5 CONSEQUÊNCIAS DO MULTIPROCESSAMENTO

Existem características do uso de multiprocessamento, a seguir serão apresentadas algumas.

1. Alta taxa de transferência: o número de processos executados em um determinado tempo é maior se diversos processadores estiverem trabalhando em conjunto
2. Como os sistemas de multiprocessadores dividem a mesma memória, periféricos e fornecimento de energia, acabam se tornando mais econômicos a longo prazo
3. Como todos os processadores compartilham a mesma memória RAM, o consumo é muito mais alto se comparado com sistemas de processadores únicos, necessitando de um conjunto de memória maior para utilizar todo o seu potencial.

Na próxima seção abordaremos mais detalhes sobre o Ysera a ferramenta que foi otimizada utilizando os conceitos de paralelismo e multiprocessamentos descritos neste trabalho

2.6 YSERA

O estudo in silico da estrutura de proteínas atua principalmente buscando melhorar metodologias de predição de contato resíduo-resíduo, tanto em metodologias de docking molecular, como de simulações de dinâmicas moleculares [NORTON, 2017]. Para tanto, é necessário o entendimento dos parâmetros físico-químicos e o cálculo de distâncias entre os átomos para definição do tipo de interação existente entre eles.

A importância de predizer essas interações fornece informações sobre a estrutura nativa de proteínas, sítio catalítico e outras propriedades físicas, sendo base para o desenvolvimento racional de fármacos e agroquímicos, engenharia de proteínas e enzimas, biologia sistêmica, entre outras [NORTON, 2017].

O Ysera é um software para predição de interações entre resíduos de aminoácidos de proteínas a partir de arquivos de estruturas tridimensionais (.PDB). Foi desenvolvido na linguagem Python v 3.7 e com bibliotecas de manipulação e análises de dados, visando calcular a distância euclidiana de todos os átomos contra todos, a fim de quantificar e qualificar interações intramoleculares[FELIX, 2020].

O Ysera é um programa de entrada e saída de dados, onde recebe como entrada um arquivo PDB que contém as informações sobre os átomos constituintes das proteínas bem como suas localizações espaciais com as coordenadas X, Y e Z. Com essas informações espaciais, calcula-se a distância euclidiana de todos os átomos em relação a todos os átomos sendo analisadas as interações encontradas. No final o programa gera um arquivo de texto contendo a contagem de interações encontradas para cada PDB. O software realiza a identificação das interações moleculares: pontes salinas, ligações dissulfeto, ligações de hidrogênio, interações Cátion- Arl, interações nion-Aril, interações Sulfeto-Aril, interações de Van Der Waals e Empilhamento π-π, quantificando-as.

Existem outras ferramentas similares ao Ysera como o Protein Interaction Calculator (PIC)[1], Arpeggio[2], um servidor da web e o O RING 2.0 (PIOVESAN; MINERVINI; TOSATTO, 2016). Todos eles não utilizam multiprocessamento e dependendo do tamanho do PDB de entrada o tempo computacional para extrair as informações desejadas pode ser muito elevado.

2.7 TRABALHOS RELACIONADOS

[1])(http://pic.mbu.iisc.ernet.in/job.html)
[2](http://biosig.unimelb.edu.au/arpeggioweb/about/)

Nesta seção serão apresentados outros trabalhos que demonstram o potencial que o multiprocessamento tem de reduzir o tempo computacional utilizando Python.

O primeiro trabalho selecionado [SINDHU, 2018] nos apresenta uma implementação simultânea do problema Max-p. O autor deste trabalho atinge velocidades de até 12 e 19 vezes com o melhor algoritmo sequencial para desenvolver e melhorar fases utilizando uma estrutura de dados multi-lock intuitiva. Para validar os algoritmos, o autor fornece resultados experimentais detalhados.

Em [KREADY, 2020] é proposta a implementação de uma aplicação paralela para a API de dados do YouTube utilizando o multiprocessamento do Python para processá-la. Os testes deste trabalho mostram que o multiprocessamento aumenta em 400% a coleta de dados do YouTube. Essas melhorias reduzem o tempo de computação por meio de utilização da arquitetura de CPU multi-thread.

Em [SILVA, 2018], assim como no nosso trabalho, os autores utilizam da biblioteca *multiprocessing* e do objeto pool, para paralelizar um script que automatiza execuções de modelos agrícolas. Como resultado obteve-se uma melhoria de desempenho em relação ao script original de aproximadamente 65%, reduzindo o tempo necessário para obter os dados de saída desejados.

Na Tabela 1, é apresentado um resumo dos trabalhos citados, com seus autores e objetivos, apresentando também o problema e o método utilizado e a área que é aplicada.

Tabela 1 — Resumo dos trabalhos relacionados

Autor	Objetivo	Método	Problema	Área aplicada
Viney Sindhu [4	Implementação paralela do problema Max-p	Python	Eficiência e Sinergia da Área Max-P	Geoespacial
J. Kready, S. A. Shimray, M. N. Hussain, and N. Agarwal [5]	Reduzir o tempo computacional da solicitação da API de dados do YouTube em paralelo	Python	O tempo que levam as solicitações da API de dados do YouTube	Multimídia
L. F. da Silva, A. S. Charao, R. P. Benedetti, N. A.	Reduzir o tempo computacional de um script que automatiza execuções de modelos agrícolas	Python	O tempo que levam os scripts de automação	Agricultura

Streck [6]				
Ferreira (2022)	Reduzir o tempo computacional de um software que calcula a interação e distância entre átomos	Python	O tempo e o custo que leva calcular a interação e distância entre átomos	Biotecnologia

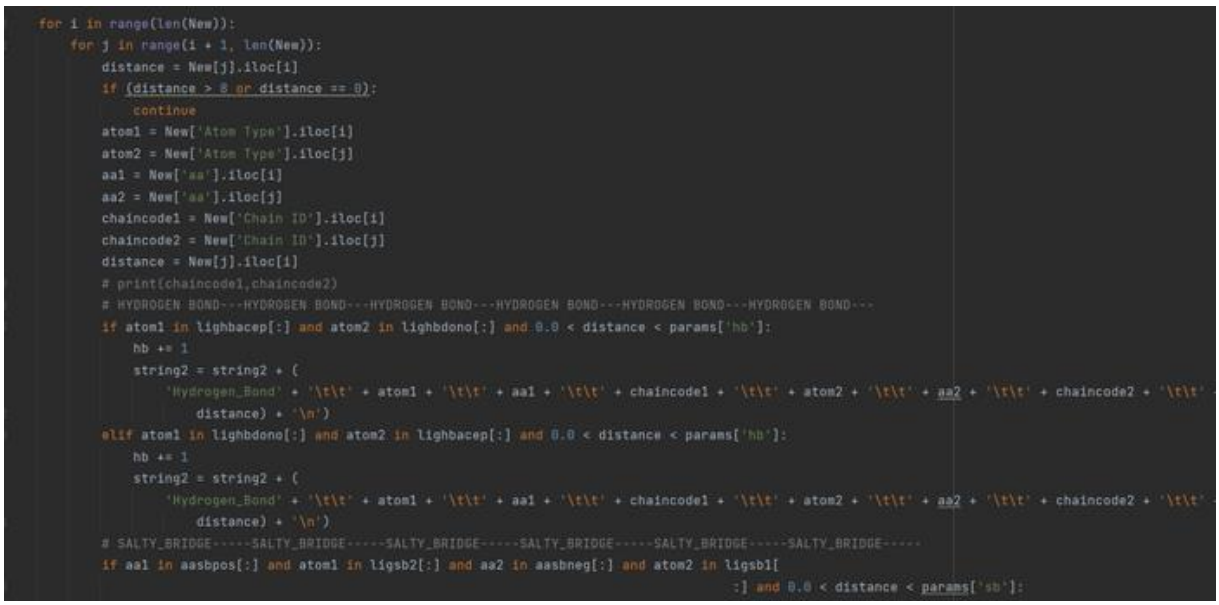
Fonte: Ferreira (2022)

A principal diferença deste trabalho para os outros que utilizam das ferramentas de multiprocessamento do Python para otimização é a área aplicada. Em [SILVA, 2018] ele utiliza exatamente da biblioteca *multiprocessing* e do objeto pool para otimizar script de execuções automáticas de modelos agrícolas. Aqui utilizamos desses mesmos meios para distribuir entre os processos partes de uma estrutura de repetição dupla e executá-la de forma paralela.

3 METODOLOGIA

A metodologia deste trabalho tem como ponto de partida a análise do código do Ysera, desenvolvido para ser executado de forma serial, e que será utilizado como referência para análise dos resultados desta pesquisa. O objetivo desta análise é entender o funcionamento do código, e se é possível otimizar sua execução através de paralelização utilizando multiprocessamento. Com essa análise, foi identificada uma estrutura de repetição dupla que recebe como índices de execução as linhas e colunas do arquivo PDB de entrada. Essa mesma estrutura de repetição é responsável pela maior parte de tempo de execução do programa, com isso ela acaba se tornando o alvo da paralelização, essa estrutura é apresentada na Figura 5

Figura 5 — Estrutura de repetição dupla, imagem tirada da versão original do Ysera.

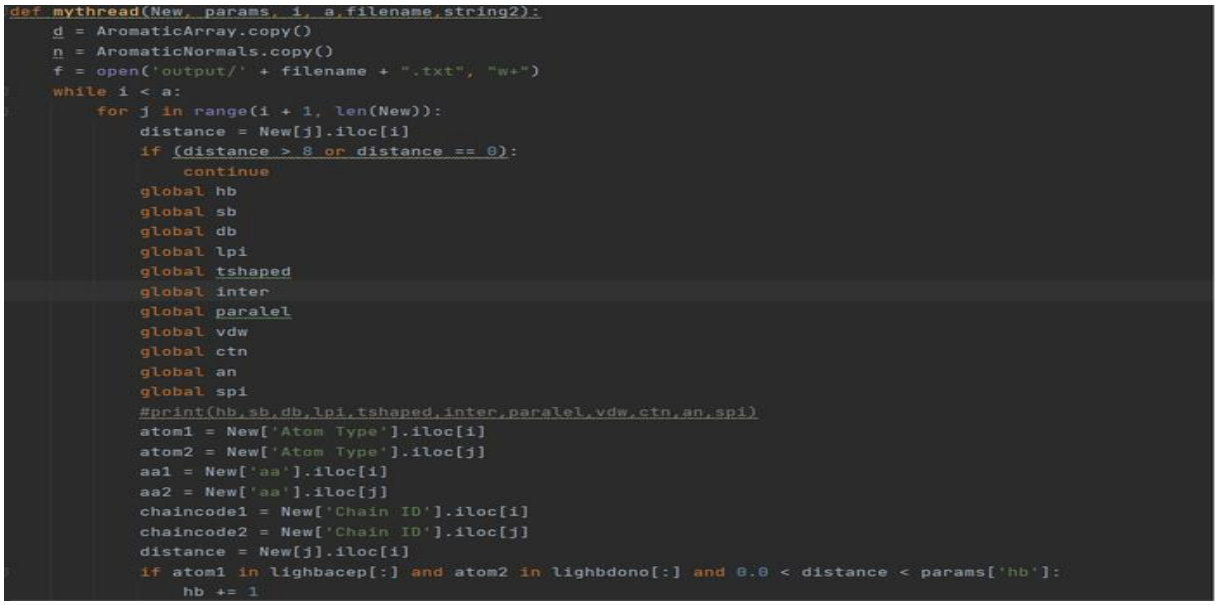


```
for i in range(len(New)):
    for j in range(i + 1, len(New)):
        distance = New[j].iloc[i]
        if (distance > 8 or distance == 0):
            continue
        atom1 = New['Atom Type'].iloc[i]
        atom2 = New['Atom Type'].iloc[j]
        aa1 = New['aa'].iloc[i]
        aa2 = New['aa'].iloc[j]
        chaincode1 = New['Chain ID'].iloc[i]
        chaincode2 = New['Chain ID'].iloc[j]
        distance = New[j].iloc[i]
        # print(chaincode1,chaincode2)
        # HYDROGEN BOND---HYDROGEN BOND---HYDROGEN BOND---HYDROGEN BOND---HYDROGEN BOND---HYDROGEN BOND---
        if atom1 in lighbacep[:] and atom2 in lighbdono[:] and 0.0 < distance < params['hb']:
            hb += 1
            string2 = string2 + (
                'Hydrogen_Bond' + '\t\t' + atom1 + '\t\t' + aa1 + '\t\t' + chaincode1 + '\t\t' + atom2 + '\t\t' + aa2 + '\t\t' + chaincode2 + '\t\t'
                    distance) + '\n')
        elif atom1 in lighbdono[:] and atom2 in lighbacep[:] and 0.0 < distance < params['hb']:
            hb += 1
            string2 = string2 + (
                'Hydrogen_Bond' + '\t\t' + atom1 + '\t\t' + aa1 + '\t\t' + chaincode1 + '\t\t' + atom2 + '\t\t' + aa2 + '\t\t' + chaincode2 + '\t\t'
                    distance) + '\n')
        # SALTY_BRIDGE-----SALTY_BRIDGE-----SALTY_BRIDGE-----SALTY_BRIDGE-----SALTY_BRIDGE-----SALTY_BRIDGE-----
        if aa1 in aasbpos[:] and atom1 in ligsb2[:] and aa2 in aasbneg[:] and atom2 in ligsb1[
                                                                                :] and 0.0 < distance < params['sb']:
```

Fonte: O autor (2022).

Foi necessário fazer alterações neste trecho do Ysera para que ele pudesse ser executado de forma paralela. A primeira delas foi transformar toda a estrutura de repetição dupla em uma função, que recebe como parâmetro partes do seu índice de execução original como pode ser visto na Figura 6.

Figura 6 — Função estruturada através da estrutura de repetição tirada da versão original do Ysera.

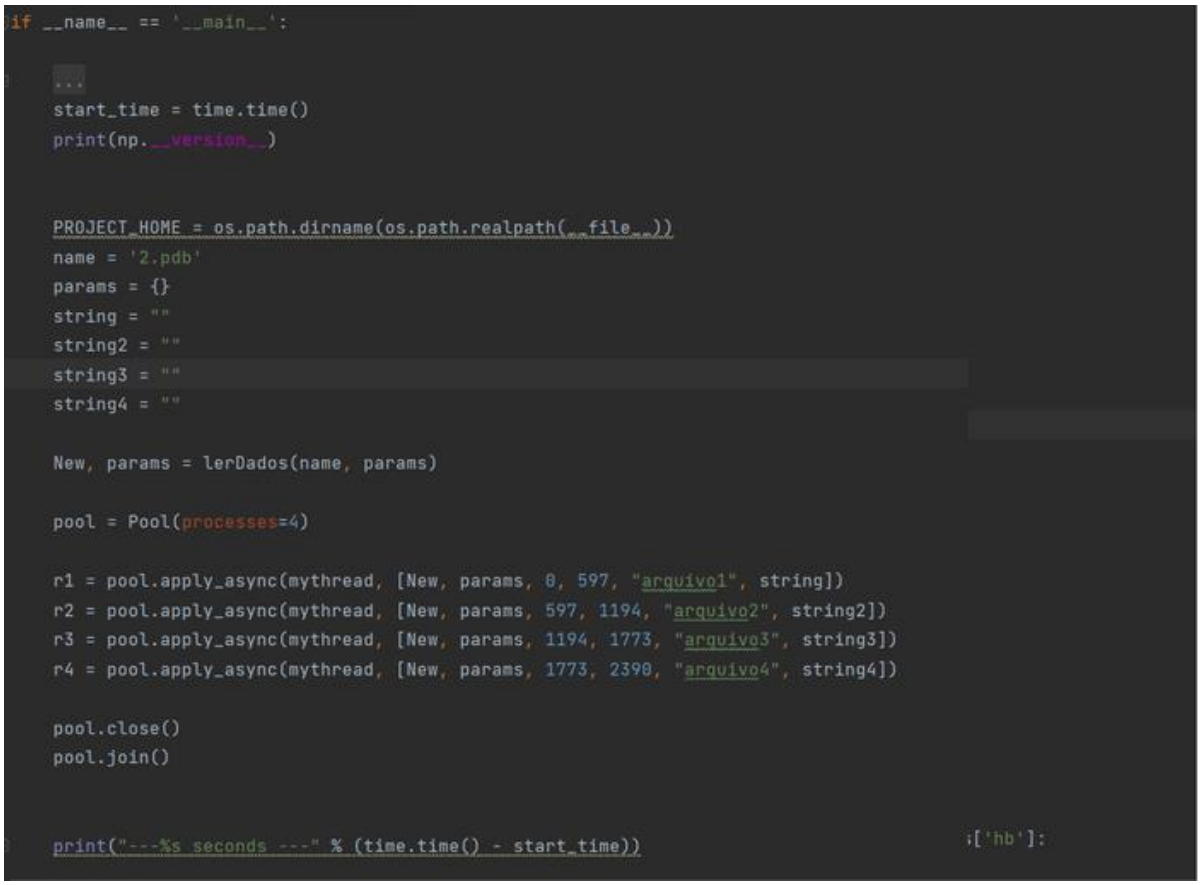


```
def mythread(New, params, i, a,filename,string2):
    d = AromaticArray.copy()
    n = AromaticNormals.copy()
    f = open('output/' + filename + ".txt", "w+")
    while i < a:
        for j in range(i + 1, len(New)):
            distance = New[j].iloc[i]
            if (distance > 8 or distance == 0):
                continue
            global hb
            global sb
            global db
            global lpi
            global tshaped
            global inter
            global paralel
            global vdw
            global ctn
            global an
            global spi
            #print(hb,sb,db,lpi,tshaped,inter,paralel,vdw,ctn,an,spi)
            atom1 = New['Atom Type'].iloc[i]
            atom2 = New['Atom Type'].iloc[j]
            aa1 = New['aa'].iloc[i]
            aa2 = New['aa'].iloc[j]
            chaincode1 = New['Chain ID'].iloc[i]
            chaincode2 = New['Chain ID'].iloc[j]
            distance = New[j].iloc[i]
            if atom1 in lighbacep[:] and atom2 in lighbdono[:] and 0.0 < distance < params['hb']:
                hb += 1
```

Fonte: O autor (2022).

A ideia principal disso é executar várias funções ao mesmo tempo, cada uma delas em um processo distinto, como exemplificado na Figura 7, a seguir.

Figura 7 — Funções sendo chamadas para serem executadas de forma paralela em um pool de processos.



```
if __name__ == '__main__':

    ...
    start_time = time.time()
    print(np.__version__)


    PROJECT_HOME = os.path.dirname(os.path.realpath(__file__))
    name = '2.pdb'
    params = {}
    string = ""
    string2 = ""
    string3 = ""
    string4 = ""

    New, params = lerDados(name, params)

    pool = Pool(processes=4)

    r1 = pool.apply_async(mythread, [New, params, 0, 597, "arquivo1", string])
    r2 = pool.apply_async(mythread, [New, params, 597, 1194, "arquivo2", string2])
    r3 = pool.apply_async(mythread, [New, params, 1194, 1773, "arquivo3", string3])
    r4 = pool.apply_async(mythread, [New, params, 1773, 2390, "arquivo4", string4])

    pool.close()
    pool.join()


    print("---%s seconds ---" % (time.time() - start_time))                    ;['hb']:
```

Fonte: O autor (2022).

Um dos primeiros passos a se fazer ao trabalhar com paralelismo em Python é passar pelas limitações que o GIL (Global Interpreter Lock) impõe. O GIL é um mutex que permite que apenas uma thread esteja em estado de execução por vez no interpretador Python [HUNT, 2019]. A ferramenta utilizada para contornar essas limitações foi a biblioteca *multiprocessing*.

Com essa biblioteca criamos uma pool de processos definida na documentação do *multiprocessing*, que trata de um objeto que oferece um meio conveniente de paralelizar a execução de uma função em vários valores de entrada, distribuindo os dados de entrada entre os processos (paralelismo de dados) [PYTHON 3 DOCUMENTATION, 2023].

Também foi necessário acrescentar um objeto do pacote *multiprocessing* chamado Manager(). Como o Ysera possui várias listas e dicionários, é necessário utilizar esse objeto para conseguir trabalhar com esse tipo de estrutura entre os processos.

3.1 MATERIAIS

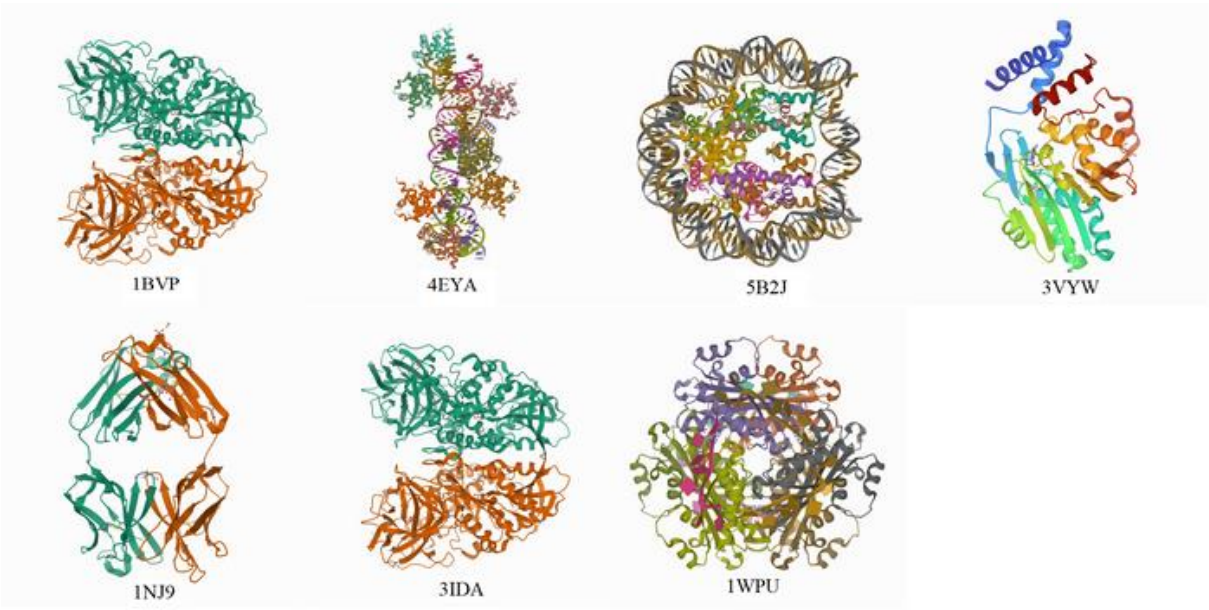
O desenvolvimento deste trabalho se deu em um ambiente da ferramenta Pycharm, que contém a linguagem de programação Python (Versão 3.6). Além disso, como dito anteriormente, foi utilizado o pacote *multiprocessing* que oferece simultaneidade local e remota, efetivamente contornando o Global Interpreter Lock usando subprocessos, em vez de threads. Devido a isso, o módulo *multiprocessing* permite que o programador aproveite totalmente vários processadores em uma determinada máquina. Ele funciona em Unix e Windows [PYTHON 3 DOCUMENTATION, 2023].

O hardware utilizado foi um notebook com um processador Intel(R) Core(TM) i7-9750H CPU & 2.60GHz, 2592 Mhz, 6 Núcleos, 12 Processadores Lógicos, com capacidade de 16 GB DDR4 de memória RAM de 1330.8MHz e 1 TB de disco rígido, Windows 11 utilizando WSL2 para executar o programa.

3.2 ARQUIVOS DE TESTE

Todos os arquivos utilizados para os testes foram retirados do RCSB PDB (https://www.rcsb.org/). No total foram 7 arquivos PDB considerando a diversidade de seus tamanhos. Na figura 8 são exibidas as estruturas tridimensionais de todos os PDBs utilizados neste trabalho.

Figura 8 — Estruturas tridimensionais dos PDBs utilizados neste trabalho geradas pelo PISA (software)



Fonte: https://www.rcsb.org/

Titulo: THE CRYSTAL STRUCTURE OF BLUETONGUE VIRUS VP7
Nome do Arquivo: 1BVP
Tamanho:[16218 Linhas x 16225 Colunas]

Titulo: CRYSTAL STRUCTURE OF A PLECTONEMIC RNA SUPERCOIL
Nome do Arquivo: 4EYA
Tamanho: [14060 Linhas x 14067 Colunas]

Titulo: HUMAN NUCLEOSOME CONTAINING CPG METHYLATED DNA
Nome do Arquivo: 5B2J
Tamanho:[12092 Linhas x 12099 Colunas]

Titulo: CRYSTAL STRUCTURE OF MNMC2 FROM AQUIFEX AEOLICUS
Nome do Arquivo: 3VYW
Tamanho: [10194 Linhas x 10201 Colunas]

Titulo: COCAINE HYDROLYTIC ANTIBODY 15A10
Nome do Arquivo:1NJ9
Tamanho: [6472 Linhas x 6479 Colunas]

Titulo: THERMOSTABLE COCAINE ESTERASE WITH MUTATIONS L169K AND G173Q, BOUND TO DTT ADDUCT
Nome do Arquivo: 3ida
Tamanho: [4866 Linhas x 4873 Colunas]

Titulo: CRYSTAL STRUCTURE OF THE HUTP ANTITERMINATION COMPLEX BOUND TO A SINGLE STRANDED REGION OF HUT MRNA
Nome do Arquivo: 1WPU
Tamanho: [2532 Linhas x 2539 Colunas]

3.3 AVALIAÇÃO

A avaliação de eficiência da paralelização do Ysera foi feita da seguinte forma: cada arquivo PDB citado anteriormente foi executado 10 vezes no total: 5 vezes de forma linear e 5 vezes de forma paralela, o objetivo das 5 execuções é calcular uma média de tempo das execuções lineares e paralelas, as execuções apresentaram resultados similares. As primeiras execuções foram feitas utilizando uma pool com apenas 4 processos, com o intuito de avaliar o quanto o tamanho do arquivo PDB influência na otimização. Dessa forma, garante-se que o mesmo número de processos para todos os arquivos será usado. Além disso, existe limitação de memória para execução do maior PDB (1BVP), onde consegue-se executar apenas com 4 processos.

Durante sua execução foram avaliados seu tempo de execução. O uso de memória RAM e o uso de processamento foi avaliado utilizando a ferramenta CPUID HWMonitor[3] v1.44. Todos esses dados foram armazenados em tabelas que serão apresentadas na próxima seção.

Foi feita ainda a avaliação de execuções com o PDB 3VYM, executando-o aumentando o número de processos e avaliando sua otimização com isso. Os valores foram normalizados e colocados em gráficos que serão apresentados a seguir, o intuito desse experimento é avaliar o quanto o número de processos influencia na otimização.

[3]https://www.cpuid.com/softwares/hwmonitor.html

4 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

Nesta seção serão apresentados os resultados decorrentes da avaliação descrita na seção anterior, sendo assim, serão apresentadas tabelas com seus respectivos dados de execução de cada arquivo PDB e também serão apresentados gráficos mostrando os efeitos do aumento do número de processos em um mesmo arquivo PDB.

Nos primeiros testes conseguiu-se uma otimização média de 43,44% de todos os PDBs em relação a versão serial, utilizando apenas 4 funções e 4 processos. O aumento do consumo de memória e processamento é notável se comparar com a versão serial do Ysera como pode ser visto nas tabelas a seguir, e esse consumo aumenta cada vez mais, à medida que as dimensões do arquivo PDB utilizado cresce. O maior PDB (1BVP) possui uma otimização de 45,93% e o menor, 41,88% (1WPU) em relação a versão serial, uma diferença de apenas 4%. Isso significa que, independente do tamanho do arquivo, o multiprocessamento é eficiente para otimizá-lo. Nas Tabelas 1 e 2, são exibidos os dados das execuções em série e em paralelo, respectivamente.

Tabela 2 — Dados das execuções em série

Nome	Tamanho	Tempo de execução	Memória	Processador
1WPU	2532 Linhas x 2539 Colunas	49,311091	35,60%	23,80%
3IDA	4866 Linhas x 4873 Colunas	161,405261	36,80%	25,20%
1NJ9	6472 Linhas x 6479 Colunas	265,722472	39,40%	33,60%
3VYW	10194 Linhas x 10201 Colunas	654,019563	41,20%	57,80%
5B2J	12092 Linhas x 12099 Colunas	866,472520	44,80%	34,60%
4EYA	14060 Linhas x 14067 Colunas	1102,164445	47,20%	76,80%
1BVP	16218 Linhas x 16225 Colunas	1598,538709	49%	34,60%

Fonte: O autor (2022)

Tabela 3 — Dados das execuções paralelas

Nome	Tamanho	Tempo de execução	Memória	Processador
1WPU	2532 Linhas x 2539 Colunas	28,660274	44,80%	41,80%
3IDA	4866 Linhas x 4873 Colunas	88,177183	47%	41%

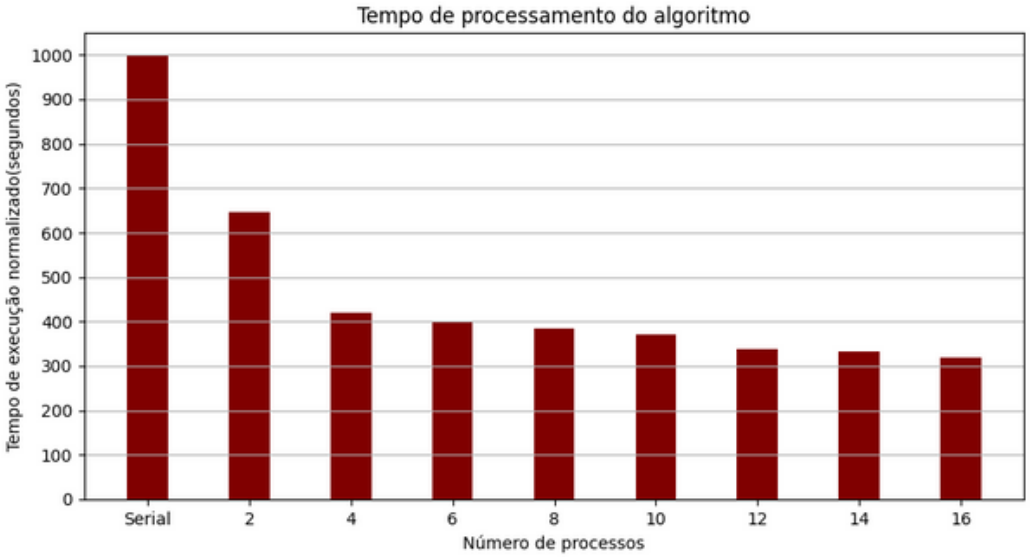
1NJ9	6472 Linhas x 6479 Colunas	150,545614	53,40%	56,80%
3VYW	10194 Linhas x 10201 Colunas	359,611580	62,60%	94,40%
5B2J	12092 Linhas x 12099 Colunas	484,469732	66%	84,20%
4EYA	14060 Linhas x 14067 Colunas	677,867997	78%	91,20%
1BVP	16218 Linhas x 16225 Colunas	864,308403	85,20%	87%

Fonte: O autor (2022)

Os testes com o aumento do número de processos utilizando o PDB 3VYW podem ser vistos na Figura 9. É possível observar que, com 4 processos, a otimização mais eficiente foi alcançada, pois, após ela, a redução do tempo computacional com o aumento do número de processos é bastante reduzida(aproximadamente 4.7% a cada 2 processos em diante). Utilizando 4 processos conseguiu-se uma otimização de 57,90%, diferente da anterior que foi de 45,11%. Como dito anteriormente, a versão paralela do Ysera funciona executando várias funções ao mesmo tempo. Cada uma dessas funções executam partes da estrutura de repetição original e são distribuídas entre os processos para operarem paralelamente. Nos testes, com o aumento do número de processos, foram criadas 16 funções, e foi-se aumentando o número de processos no decorrer dos testes. Isso significa que aumentar o número de funções, também é uma forma de otimização. Quanto mais funções distribuídas, maior é o consumo de memória.

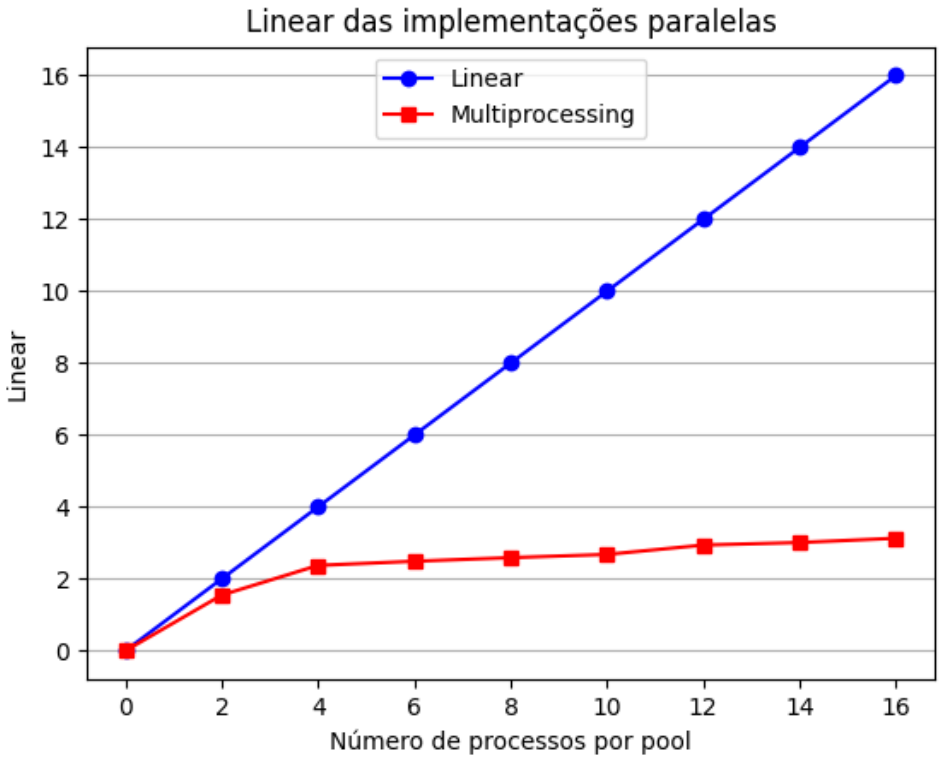
Na Figura 10 observa-se o gráfico Linear, que demonstra que a eficiência de otimização diminui com o aumento do número de processos. Os testes foram feitos com até 16 processos pela limitação de memória. No entanto, o aumento contínuo do número de processos poderia acarretar em sobrecarga, que pode ser gerada na fila de escalonamento do processador [SILVA, 2018]. Isso acontece quando o número de processos colocados para execução é maior que o número de cores, tornando a execução mais lenta do que a anterior, com menos processos.

Figura 9 — Tempos de execução com o numero de processos por pool.



Fonte: O autor (2022)

Figura 10 — Gráfico Linear obtido com as implementações paralelas. Onde a linha linear significa a otimização perfeita e o multiprocessing a otimização alcançada neste trabalho.



Fonte: O autor (2022).

5 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Conseguindo diminuir o tempo de execução mais do que pela metade, o pacote *multiprocessing* do Python revelou ser uma solução viável para a otimização do cálculo das distâncias e interações entre átomos. A principal limitação da versão paralela do Ysera é que se não existe memória suficiente, o programa não executa, diferente da versão serial do Ysera que, mesmo quando utiliza um PDB que extrapola o uso de memória, ele ainda vai funcionar, mesmo com um tempo maior. Essa limitação da versão paralela pode ser contornada com um bom gerenciamento do número de processos. Como visto anteriormente, apenas com 4 já se consegue uma otimização eficiente, ou simplesmente com mais hardware.

5.1 TRABALHOS FUTUROS

Como trabalhos futuros, a versão paralela do Ysera funciona de forma manual, o usuário divide o número de funções e processadores. Uma automatização da divisão de funções e processos calculando o tamanho do arquivo de entrada e a memória disponível seria uma grande melhoria, pois ele sempre executaria qualquer PDB na maior velocidade possível.

REFERÊNCIAS

FARIA, Gustavo. **Paralelismo OU**. Disponível em: https://www.cos.ufrj.br/~ines/courses/LP/98-1/Paralelismo.htm. Acesso em: 27 jun. 2023.

FELIX, YURI. Desenvolvimento de software para análise de interações atômicas de proteínas com mutações deletérias e não deletérias para a previsão de câncer. **Preditor De Interações Atômicas De Estruturas Tridimensionais De Proteínas**, 2020.

FERREIRA, Alielson. Desenvolvimento de interface web para um software para predição de interações entre resíduos de aminoácidos em um polipeptídeo. **Preditor De Interações Atômicas De Estruturas Tridimensionais De Proteínas**, 2020.

HUNT, John. **Advanced Guide to Python 3 Programming**. Springer Nature, f. 249, 2019. 497 p.

KREADY, Joseph *et al*. 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). **YouTube data collection using parallel processing**, 2020.

NAVARRO, Cristóbal; HITSCHFELD-KAHLER, Nancy; MATEU, Luis. Communications in Computational Physics, 15(2), 285-329. doi:10.4208/cicp.110113.010813. **A Survey on Parallel Computing and its Applications in Data-Parallel Problems Using GPU Architectures.**, 2014.

NORTON, Raymond. Expert Opinion On Drug Discovery. **Enhancing the therapeutic potential of peptide toxins**, 20 abr 2017.

PIKE, Rob. **"Concurrency is not parallelism @ Heroku's Waza Conference."**. Disponível em: https: //vimeo.com/49718712. Acesso em: 27 jun. 2023.

PYTHON 3 DOCUMENTATION, Python Docs (2022). Disponível em: https://docs.python.org/3/library/multiprocessing.html. Acesso em: 27 jun. 2023.

SILBERSCHATZ, Abraham; GALVIN, Peter B.; GAGNE, Greg. **Fundamentos de sistemas operacionais**. 9 ed. LTC, 2015.

SILVA, Lucas *et al*. ESCOLA REGIONAL DE ALTO DESEMPENHO DA REGIÃO SUL . **Explorando Paralelismo em Python para Múltiplas Execuções de Modelos de Culturas Agrícolas**, Porto Alegre, 2018.

SINDHU, Viney. **Exploring Parallel Efficiency and Synergy for Max-P Region Problem Using Python**. 2018 Tese - Georgia State University.

TANENBAUM, Andrew S.; BOS, Herbert. **Sistemas operacionais modernos**. 4 ed. Pearson, 2016.