Vinícius de Azevedo Menezes

Implementando recursos de programação paralela e analisando comportamento de plataforma de correção de posições corporais em exercícios físicos

Natal – RN

Dezembro de 2023

Vinícius de Azevedo Menezes

Implementando recursos de programação paralela e analisando comportamento de plataforma de correção de posições corporais em exercícios físicos

Trabalho de Conclusão de Curso de Engenharia de Computação da Universidade Federal do Rio Grande do Norte, apresentado como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação

Orientador: Samuel Xavier de Souza

Universidade Federal do Rio Grande do Norte – UFRN

Departamento de Engenharia de Computação e Automação – DCA

Curso de Engenharia de Computação

Natal – RN

Dezembro de 2023

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Menezes, Vinicius de Azevedo.
Implementando recursos de programação paralela e analisando comportamento de plataforma de correção de posições corporais em exercícios físicos / Vinicius de Azevedo Menezes. - 2023.
47 f.: il.

Monografia (graduação) - Universidade Federal do Rio Grande do Norte, Centro de Tecnologia, Curso de Engenharia de Computação e Automação, Natal, RN, 2023.
Orientação: Prof. Dr. Samuel Xavier de Souza.

1. Programação paralela - Monografia. 2. Reconhecimento corporal - Monografia. 3. Exercícios físicos - Monografia. 4. Arquitetura em nuvem - Monografia. 5. Otimização de desempenho - Monografia. I. Souza, Samuel Xavier de. II. Título.

RN/UF/BCZM CDU 004.2

Elaborado por Ana Cristina Cavalcanti Tinoco - CRB-15/262

Vinícius de Azevedo Menezes

Implementando recursos de programação paralela e analisando comportamento de plataforma de correção de posições corporais em exercícios físicos

Trabalho de Conclusão de Curso de Engenharia de Computação da Universidade Federal do Rio Grande do Norte, apresentado como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação

Orientador: Samuel Xavier de Souza

Trabalho aprovado. Natal – RN, 12 de Dezembro de 2023, Banca Examinadora:

Prof. Dr. Samuel Xavier de Souza - Presidente da banca
UFRN

Prof. Dr. Agostinho De Medeiros Brito Junior - Examinador
UFRN

Prof. Dr. Ivanovitch Medeiros Dantas Da Silva - Examinador
UFRN

Natal – RN
Dezembro de 2023

AGRADECIMENTOS

Agradeço primeiramente a meus pais, por todo o esforço para este momento chegar em minha vida.

Agradeço a minha esposa, que me deu todo o suporte para conseguir finalizar esse projeto.

Agradeço também aos meus colegas da faculdade, que foram essenciais nessa longa jornada de conhecimentos.

Agradeço aos professores e ao meu orientador Samuel Xavier de Souza pelo auxílio e conselhos durante a execução deste trabalho.

"Feliz o homem que encontrou a sabedoria e alcançou o entendimento,
porque a sabedoria vale mais do que a prata,
e dá mais lucro que o ouro."
(Bíblia Sagrada, Provérbios 3, 13-14)

RESUMO

Este Trabalho de Conclusão de Curso tem como objetivo geral a aplicação estratégica de recursos de Programação Paralela para resolver problemas de desempenho em uma aplicação de reconhecimento e correção de posições corporais durantes exercícios físicos através da coleta de imagens via webcam. A aplicação requer uma rápida coleta de dados do usuário em tempo real e a entrega ágil do resultado processado na tela da plataforma. Por consequência, também são gerados desafios relacionados ao armazenamento e tráfego de dados na web, essenciais para a comparações e cálculos necessários. Os objetivos do trabalho são focados em analisar o comportamento da plataforma em relação ao usuário final, analisar gargalos de desempenho de código por meio de técnicas como perfilagem e aprimorar o código utilizando técnicas de paralelização;

Palavras-chaves: Programação Paralela; Reconhecimento Corporal; Exercícios Físicos; Arquitetura em Nuvem; Otimização de Desempenho.

ABSTRACT

This Undergraduate Thesis aims at the overall application of Parallel Programming resources strategically to address performance issues in a platform for recognition and correction of body positions during physical exercises through the collection of images via webcam. The application requires a swift real-time data collection from the user and the prompt delivery of the processed result on the platform's screen. Consequently, challenges related to data storage and web traffic are also generated, crucial for necessary comparisons and calculations. The objectives of the study are focused on analyzing the platform's behavior concerning the end user, examining performance bottlenecks in the code through techniques such as profiling, and enhancing the code using parallelization techniques.

Keywords: Parallel Programming; Body Recognition; Physical Exercises; Cloud Architecture; Performance Optimization.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

JSON	*JavaScript Object Notation*
API	*Application Programming Interface*
RAM	*Random Access Memory*

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização do problema

Atualmente, é notável o crescente emprego de diversas tecnologias que permeiam e impactam significativamente nosso cotidiano, independentemente de serem perceptíveis ou operarem nos bastidores. O surgimento constante de inovações tecnológicas, como computação quântica, internet das coisas, inteligência artificial, realidade aumentada e diversas outras, contribui para a transformação acelerada do panorama tecnológico. Este fenômeno revela a onipresença da tecnologia na sociedade atual, desempenhando um papel fundamental em diferentes aspectos da vida diária. Essas inovações também são aplicadas à plataforma objeto de estudo no presente trabalho, trata-se de uma plataforma desenvolvida com o propósito de realizar correções em tempo real nos movimentos executados durante exercícios físicos através da captura de imagens via webcam, simulando a presença de um profissional ao lado do usuário. Para viabilizar essa aplicação, são empregadas técnicas de Álgebra Linear, Matemática, processamento de imagens, computação em nuvem, entre outras tecnologias.

A plataforma em questão, é disponibilizada exclusivamente para educadores físicos ou fisioterapeutas, que, por sua vez, orientam o uso da mesma para seus pacientes ou alunos. Dessa forma, se a aplicação apresentar correções incorretas ou um desempenho abaixo do esperado, há o risco de agravar situações de pacientes em processo de reabilitação fisioterapêutica ou causar lesões nos praticantes de exercícios devido a feedbacks imprecisos. Portanto, a integridade operacional da plataforma é de extrema prioridade.

Para compreender a arquitetura subjacente a essa aplicação, é imperativo explorar os seguintes macro passos na figura 1.

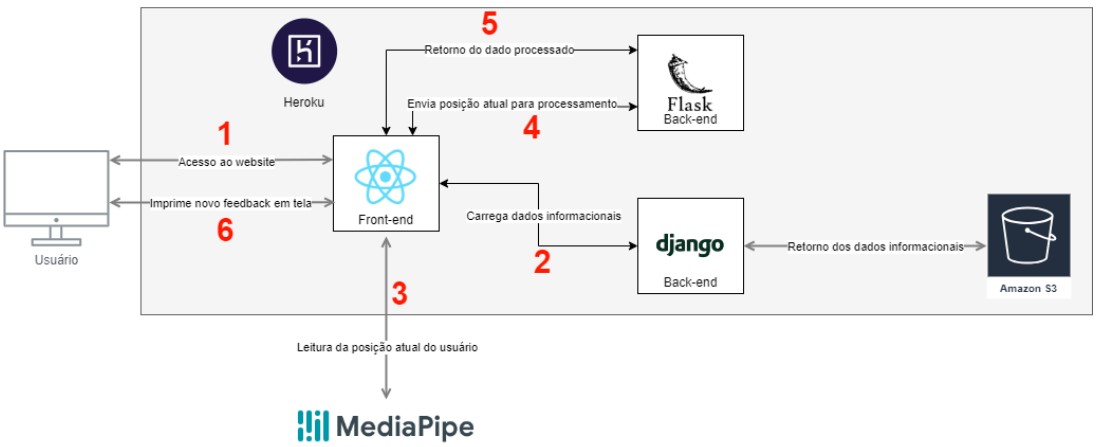

Figura 1 – Arquitetura da plataforma de correção de posições corporais.

1. O usuário acessa ao website;
2. A aplicação, por sua vez, consulta os dados informativos do treino do usuário na nuvem e os exibe na tela;
3. A leitura dos movimentos se inicia, por meio da webcam do usuário;
4. A plataforma capta e registra a posição atual do usuário;
5. Realiza-se uma comparação entre a posição atual do usuário e a posição correta estipulada pelo profissional;
6. O feedback gerado a partir da comparação anterior é exibido na tela;
7. Esse ciclo se repete várias vezes até a conclusão do exercício;

Um importante destaque nesta etapa é entender como as articulações são lidas a partir da imagem da webcam do usuário. Essa tarefa é realizada por uma inteligência artificial suportada pela Google chamada "MediaPipe", como podemos na etapa 3 da Figura 1. Durante o movimento do usuário, essa inteligência artificial é responsável por enviar para a plataforma as novas posições de cada articulação de acordo com o exercício. Essa é uma ferramenta de código aberto, que permite seu uso comercial de forma gratuita (DEVELOPERS, 2023).

Nesta mesma etapa, um aspecto crucial é a frequência de coleta de dados provenientes do MediaPipe. Isso é essencial para avaliar se algum dado está sendo perdido devido à falta de tempo para ser exibido na tela, visto que a próxima coleta já ocorre, resultando na sobreposição dessa informação e na recalculação com base nos dados mais recentes.

Outro ponto crítico de análise situa-se entre as etapas 4 e 5 da Figura 1: qual é o tempo necessário para processar os dados recebidos? Esse processo pode ser otimizado? Um tempo de processamento prolongado pode, inclusive, ocasionar filas de feedbacks a serem exibidos ou resultar na perda de feedbacks durante o percurso para exibição de um feedback mais atualizado.

Na etapa 6 da Figura 1, é imperativo observar também o tempo necessário para exibir o feedback na tela, levando em consideração que o usuário está acessando e utilizando uma plataforma hospedada na nuvem. Existe alguma análise de desempenho que possa levantar possíveis melhorias nessa etapa?

Diante do exposto, idealizou-se essa pesquisa para responder: Como utilizar recursos de programação paralela para resolver problemas de coleta e tratamento de dados em uma plataforma dedicada ao reconhecimento e correção de posições corporais durante exercícios físicos através da coleta de imagens via webcam?

1.2 Trabalhos Relacionados

1.2.1 Entendendo a Programação Paralela e Computação em Nuvem

De acordo com (PACHECO, 2011), até o início dos anos 2000, o desempenho dos microprocessadores aumentava, em média, 50% de um ano para outro. A consequência disso é que usuários e desenvolvedores de software podiam simplesmente esperar que, a cada 2 anos, suas aplicações teriam o dobro de performance nos novos computadores e microprocessadores. Porém, as grandes fabricantes perceberam que, ao longo dos anos, teríamos problemas de superaquecimento dos computadores devido ao transistores que existiam dentro dos circuitos. À medida que os transistores diminuem de tamanho, sua velocidade intrínseca pode ser aumentada devido a características físicas. O menor tamanho permite que os sinais eletrônicos percorram distâncias mais curtas, resultando em tempos de trânsito mais rápidos. Isso, por sua vez, contribui para o aumento da velocidade geral do circuito integrado.

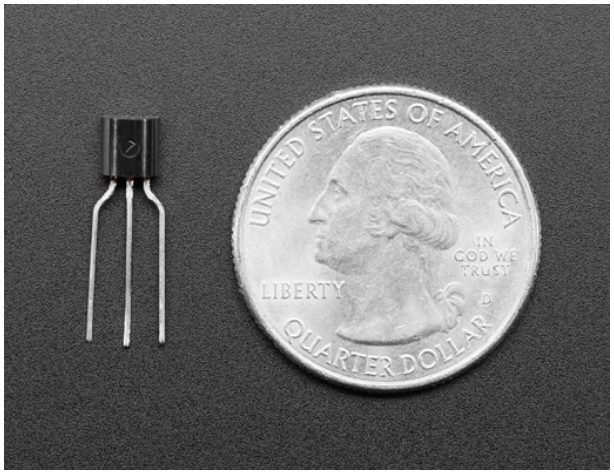

Figura 2 – *I*magem de um transitor relativamente grande comparado a uma moeda.

Fonte: (INDUSTRIES, 2017)

No entanto, é crucial notar que, à medida que a velocidade dos transistores aumenta por conta de seu tamanho reduzido, ocorre um aumento de densidade desses componentes, acarretando em um maior consumo de energia. Este consumo está associado a uma maior dissipação de energia na forma de calor. Quando a temperatura de um circuito integrado atinge níveis elevados, a confiabilidade do dispositivo pode ser comprometida, pois o superaquecimento pode afetar adversamente seu desempenho.

Diante disso, em 2005, a maioria dos principais fabricantes de microprocessadores decidiu que o caminho para aumentar rapidamente o desempenho estava na direção do paralelismo. Em outras palavras, seria colocar múltiplos processadores completos em um único circuito integrado ao invés de continuar desenvolvendo processadores monolíticos cada vez mais rápidos.

Essa alteração possui uma implicação de extrema relevância para os desenvolvedores de software, pois a simples adição de mais processadores não resultará automaticamente em melhorias significativas no desempenho da grande maioria dos programas sequenciais, isto é, programas originalmente projetados para operar em um único processador. Esses programas não têm consciência da presença de múltiplos processadores, e, consequentemente, o desempenho de um programa desse tipo em um sistema com vários processadores pode não ter a performance esperada.

Por outro lado, mas não distante, temos o surgimento do conceito de "Computação em Nuvem". Porém, ao contrário do que muitos pensam, essa não é uma ideia nova. De acordo com (ERL RICARDO PUTTINI, 2013), a ideia de computação em uma "nuvem" remonta às origens da computação utilitária, um conceito que o cientista da computação John McCarthy propôs publicamente em 1961:

> *[...] Se os computadores do tipo que defendo se tornarem os computadores do futuro, então a computação poderá algum dia ser organizada como um serviço público, tal como o sistema telefónico é um serviço público. ... A computação utilitária pode se tornar a base de uma nova e importante indústria.*

Um pouco depois, em 1969, Leonard Kleinrock, cientista-chefe da Rede de Agências de Projetos de Pesquisa Avançada ou projeto ARPANET que semeou a Internet, afirmou:

> *[...] A partir de agora, as redes de computadores ainda estão na sua infância, mas à medida que crescem e se tornam sofisticadas, provavelmente veremos a disseminação de "utilitários de computador".*

No final da década de 1990, a Salesforce.com foi pioneira na ideia de trazer serviços provisionados remotamente para a empresa. Em 2002, a Amazon.com lançou a plataforma Amazon Web Services (AWS), um conjunto de serviços voltados para empresas que fornecem armazenamento provisionado remotamente, recursos de computação e funcionalidades de negócios. Mas, foi somente em 2006 que o termo "Computação em Nuvem" surgiu na arena comercial. Isso ocorreu quando a Amazon lançou seus serviços Elastic Compute Cloud (EC2), que permitiram às organizações "alugar" capacidade computacional e poder de processamento para executar seus aplicativos empresariais. O Google Apps também começou a fornecer aplicativos empresariais baseados em navegador no mesmo ano e, três anos depois, o Google App Engine se tornou outro marco histórico.

A computação em nuvem proporciona benefícios significativos, como a redução de investimentos e custos proporcionais. Ao adotar esse paradigma, as organizações podem usufruir de infraestrutura robusta sem a necessidade de adquiri-la, pois os provedores de nuvem obtêm recursos de TI em grande escala e os disponibilizam por meio de atrativos pacotes de locação. Isso resulta na eliminação ou minimização dos gastos iniciais em

hardware e software, permitindo que as empresas comecem de maneira mais modesta e aumentem a alocação de recursos de TI conforme necessário.

Adicionalmente, destaca-se o benefício da escalabilidade, no qual as nuvens possibilitam a alocação instantânea e dinâmica de recursos de TI aos consumidores, seja sob demanda ou por meio de configuração direta. Essa capacidade estimula os consumidores de nuvem a ajustar automaticamente ou manualmente seus recursos de TI baseados em nuvem para lidar com flutuações e picos de processamento. A característica intrínseca e integrada de fornecer níveis flexíveis de escalabilidade aos recursos de TI está diretamente relacionada ao benefício de custos proporcionais mencionado anteriormente. Além do ganho financeiro evidente pela redução automatizada da escala, a habilidade dos recursos de TI de atender constantemente às demandas de uso imprevisíveis evita possíveis perdas de negócios que podem ocorrer quando os limites de uso são atingidos.

Outro ponto relevante é o aumento da disponibilidade e confiabilidade dos recursos de TI. A disponibilidade e confiabilidade desses recursos estão diretamente ligadas a benefícios tangíveis para os negócios, pois as interrupções limitam o tempo em que um recurso de TI pode estar "aberto para negócios"para seus clientes, reduzindo seu uso e potencial de geração de receita. Falhas durante a execução, se não corrigidas imediatamente, podem ter um impacto mais significativo durante períodos de uso intenso, prejudicando a confiança do cliente.

Apesar da evidente vantagem, calcular e avaliar a economia real da computação em nuvem pode ser complexo. A estratégia de adoção da nuvem envolve considerações que vão além de uma simples comparação entre os custos de locação e compra, incluindo uma análise detalhada da escalabilidade dinâmica e da transferência de riscos relacionados à sub ou superutilização de recursos. Outro aspecto crucial é que as organizações examinem minuciosamente os Acordos de Nível de Serviço (SLAs) oferecidos pelos provedores de nuvem ao considerar a locação de serviços e recursos de TI baseados em nuvem. Embora muitos ambientes de nuvem possam oferecer níveis notavelmente altos de disponibilidade e confiabilidade, isso depende das garantias estabelecidas no SLA, representando as obrigações contratuais efetivas.

No lado da Programação Paralela, a maioria dos programas desenvolvidos para sistemas convencionais single core não consegue aproveitar a presença dos multi cores. É possível executar várias instâncias de um programa em um sistema multicore, mas isso muitas vezes tem pouco benefício. Por exemplo, poder executar várias instâncias da aplicação de correção de posições corporais, focada neste trabalho, não é exatamente o que se necessita, é preciso que o programa funcione de forma mais eficiente. Para alcançar isso, precisamos reescrever ou adaptar partes sequenciais de código para torná-los paralelos, de modo que possam aproveitar os múltiplos núcleos dos processadores atuais.

1.3 Motivação

A fundamentação motivacional para este trabalho deriva da perspectiva de aplicar, de maneira prática, o conhecimento adquirido ao longo do curso de Engenharia de Computação a fim de abordar desafios tangíveis no âmbito de uma empresa em processo de criação. Empresa essa, no qual o autor deste trabalho é um dos fundadores, e que tem como objetivo melhorar a qualidade de vida da sociedade, proporcionando uma oportunidade de ter a experiência de ter um profissional de educação física ou fisioterapia ao seu lado, independente de momento ou local. Dessa forma, a plataforma acaba servindo para pessoas que não tem a oportunidade de ter acompanhamento durante exercícios físicos, sejam idosos, jovens ou pacientes em recuperação fisioterapeutica.

Além disso, é uma oportunidade para ampliar entendimento nos campos específicos de programação paralela e computação em nuvem, áreas pelas quais são imprescindíveis para o âmbito profissional de um engenheiro de computação.

1.4 Objetivos

1.4.1 Objetivo Geral

Resolver problemas de coleta e tratamento de dados em uma plataforma dedicada ao reconhecimento e correção de posições corporais durante exercícios físicos através da coleta de imagens via webcam.

1.4.2 Objetivos específicos

- Analisar o comportamento da plataforma em relação ao usuário final, como mostram as etapas 3 e 6 da Figura 1;
- Analisar gargalos de desempenho de código das etapas 4 e 5, mostradas na Figura 1, por meio de técnicas como perfilagem;
- Aprimorar o código, responsável pelo processamento de dados, utilizando técnicas de paralelização;

1.5 Estrutura do Trabalho

Este trabalho apresenta uma introdução sobre o tema, mostrando os fatores que motivam a implantação da ideia, além da justificativa e dos objetivos. Em sequência, o Capítulo 2 apresenta as ferramentas utilizadas da plataforma, desde programação paralela e computação em nuvem até linguagens de programação. O Capítulo 3, por sua vez, explica

a metodologia para implementação e alcance de todos os objetivos específicos, enquanto o Capítulo 4 trata da análise do comportamento da plataforma em nuvem e perfilagem. O Capítulo 5 apresenta a paralelização do código e os resultados alcançados. Por fim, o Capítulo 6 traz as principais conclusões e contribuições deste trabalho.

2 CAPÍTULO 2

Neste capítulo, são apresentados as ferramentas utilizadas para execução do presente estudo, desde programação paralela e computação em nuvem até as linguagens de programação.

2.1 *Linguagens de programação*

A linguagem Python, conhecida por sua sintaxe clara e elegante, emerge como uma escolha assertiva para o desenvolvimento dessa plataforma. Sua versatilidade e legibilidade facilitam a implementação de algoritmos complexos, enquanto a vasta gama de bibliotecas, como NumPy e OpenCV, oferece recursos robustos para processamento de dados e visão computacional. Esta linguagem é responsável pelo back-end da plataforma.

Por outro lado, a linguagem JavaScript desempenha um papel vital na interatividade da plataforma, proporcionando dinamismo e responsividade às interfaces de usuário. Com sua execução no lado do cliente, JavaScript permite uma experiência fluida para os usuários. Esta linguagem é responsável pelo front-end da plataforma.

A integração harmoniosa entre Python e JavaScript é facilitada pelo framework Flask. Este framework leve e modular proporciona uma estrutura robusta para o desenvolvimento de aplicações web. Sua arquitetura minimalista, aliada à extensibilidade, permite a criação eficiente de serviços web para a correção de posições corporais durante os exercícios.

2.2 *Computação em nuvem*

Quando um usuário acessa um website hospedado em nuvem, o processo inicia-se com a requisição do navegador. A solicitação é encaminhada para o servidor na nuvem, que armazena os recursos da aplicação. Em seguida, o servidor processa a requisição e envia os dados necessários de volta ao navegador do usuário, permitindo a renderização e interação com a página web.

No contexto de uma plataforma que realiza coletas por segundo através da webcam do usuário, o processo de hospedagem em nuvem pode ter implicações cruciais. O tempo de resposta do servidor, a largura de banda disponível e a capacidade de processamento influenciam diretamente a eficiência da coleta de dados em tempo real. A latência na transmissão e processamento desses dados pode comprometer a experiência do usuário e a precisão das informações coletadas.

No contexto do presente estudo, é relevante abordar que as configurações de hardware na máquina que hospeda o website ficarão fixas com 2 processadores e 1 Gigabyte de memória RAM, até o final do trabalho. Essa escolha é motivada por limitações financeiras e prioridades definidas pelo momento atual da empresa que sustenta a plataforma.

2.3 *Perfilagem*

A perfilagem de código refere-se ao processo de medição e análise do desempenho de um programa, identificando áreas de código que consomem mais recursos computacionais. Essa prática visa otimizar a eficiência do código, revelando quais partes contribuem significativamente para o tempo de execução. No contexto deste estudo, foram empregadas ferramentas específicas para a perfilagem, incluindo cProfile e pstats, ambas integradas à biblioteca padrão do Python (FOUNDATION, 2022). Além disso, o SnakeViz, uma ferramenta de visualização amigável, foi utilizada para interpretar e apresentar os resultados da análise de desempenho, facilitando a identificação de áreas críticas e a tomada de decisões para otimização (DAVIS, 2023).

2.4 *Programação Paralela*

A programação paralela é uma abordagem que busca otimizar o desempenho computacional ao realizar simultaneamente múltiplas tarefas em diferentes núcleos de processamento. Esse paradigma visa superar as limitações da programação sequencial, permitindo a execução concorrente de operações e, assim, acelerar o processamento de grandes conjuntos de dados.

O OpenMP, Open Multi-Processing, é um conjunto de diretivas e funções de API (Interface de Programação de Aplicações) destinadas a facilitar a programação paralela em linguagens como C, C++, e Fortran. Amplamente utilizado, o OpenMP simplifica a criação de código paralelo ao oferecer uma abordagem pragmática e de fácil implementação, tornando-se uma escolha popular para desenvolvedores que buscam melhorar a eficiência de seus programas.

As diretivas OpenMP são comandos especiais incorporados ao código-fonte que instruem o compilador a paralelizar determinadas seções do código. A pragma parallel for é uma diretiva específica do OpenMP que indica ao compilador para dividir um loop em iterações independentes, permitindo que diferentes threads processem simultaneamente essas iterações. Essa estratégia é particularmente eficaz em loops que realizam tarefas repetitivas, proporcionando uma distribuição eficiente do trabalho entre os núcleos de processamento disponíveis e melhorando significativamente o desempenho computacional.

Ao paralelizar código, é crucial estar atento a potenciais condições de corrida, que ocorrem quando duas ou mais threads tentam acessar e modificar os mesmos dados simultaneamente, levando a resultados imprevisíveis. Para mitigar esse problema, é essencial utilizar mecanismos de sincronização para controlar o acesso concorrente a recursos compartilhados. Além disso, deve-se evitar dependências implícitas entre threads, garantindo que dados críticos sejam manipulados de maneira atômica ou, quando possível, optando por estratégias que minimizem a necessidade de compartilhamento. Outras preocupações incluem o balanceamento de carga entre as threads para evitar ociosidade e garantir eficiência, além da consideração de possíveis efeitos colaterais, como aumento de consumo de memória. Uma abordagem cuidadosa, juntamente com testes rigorosos, é imperativa para assegurar a correta execução do código paralelo, evitando erros sutis que possam comprometer a estabilidade e a precisão do programa.

No âmbito deste estudo, a implementação de programação paralela foi efetuada por meio da utilização do OpenMP. Contudo, devido à natureza da linguagem Python, foi necessário adotar estratégias específicas para incorporar o OpenMP ao código. Para contornar essa questão, recorreu-se à importação de módulos de código em C diretamente no Python, possibilitando a integração com o OpenMP. Essa abordagem permitiu tirar proveito das funcionalidades de paralelização oferecidas pelo OpenMP, mesmo em um contexto em que a linguagem principal não tinha suporte nativo para essa ferramenta. Adicionalmente, adotou-se a estratégia de criar módulos de código em Python, promovendo uma abordagem modular e facilitando a manutenção e expansão do código ao longo do estudo.

Outra ferramenta utilizada foi o GCC (GNU Compiler Collection), que é um conjunto de compiladores de código aberto desenvolvido pelo Projeto GNU. Amplamente utilizado, suporta várias linguagens de programação, incluindo C, C++, e Fortran, compilando o código-fonte em binários executáveis para diferentes arquiteturas de hardware.

2.5 *Conhecimentos complementares*

A ferramenta DevTools do Google Chrome é uma suíte integrada de desenvolvimento web que oferece recursos abrangentes para inspecionar, depurar e otimizar páginas web. Neste estudo, a ferramenta DevTools foi empregada como uma aliada para a análise de comportamento da plataforma (DEVELOPERS, 2017). Através da inspeção dos recursos carregados, do monitoramento das requisições de rede e da identificação de gargalos de desempenho, o DevTools proporcionou insights para a compreensão do fluxo de execução da plataforma.

3 CAPÍTULO 3

Neste capítulo, é abordado sobre a metodologia para avaliação e implementação das melhorias estipuladas, como podemos ver abaixo:

- Análise da comunicação da plataforma com usuário;
- Análise do ciclo de coleta de dados;
- Passos para a paralelização de código;
- Amostra e testes.

3.1 Analisando a comunicação da plataforma com o usuário

Um importante aspecto que se buscou entender foi na comunicação entre a plataforma, e a tela do usuário. Uma arquitetura mal otimizada pode ocasionar atrasos na entrega e recebimento de mensagens e, consequentemente, a perda de dados. Portanto, torna-se essencial realizar testes de tempo de resposta após o retorno do dado tratado ao front-end, e testes de tempo de envio de novas imagens para serem processadas, a fim de identificar eventuais atrasos significativos que demandem intervenção.

3.2 Analisando o ciclo de coleta de dados

A fim de compreender o ciclo de coleta de dados do movimento do usuário durante o exercício e enviá-lo para o back-end, é essencial não apenas avaliar a frequência de coleta, mas também compreender se o tempo entre as coletas de dados é inferior ao tempo de processamento. Nesse contexto, o dado coletado do usuário pode estar sendo recebido com maior frequência do que pode ser eficientemente processado, o que resultaria em potencial perda de dados.

3.3 Passos para a paralelização de código

3.3.1 Perfilagem do Código: Identificação de Gargalos

O primeiro passo consiste na aplicação de técnicas de perfilagem no código-fonte existente. Essa abordagem permite a análise do comportamento do programa em execução,

identificando áreas suscetíveis a gargalos de desempenho. Utilizando ferramentas especializadas, serão capturadas informações detalhadas sobre o tempo de execução e o consumo de recursos em diferentes partes do código.

Durante esta fase, busca-se compreender a distribuição de carga computacional, identificando pontos críticos que impactam significativamente o tempo total de execução. A análise resultante orienta a seleção de áreas específicas para aplicação das técnicas de paralelização, garantindo um investimento estratégico de esforços na otimização do desempenho.

Adicionalmente, é de significativa importância considerar de maneira aprofundada os resultados provenientes da perfilagem, a fim de verificar a pertinência da eventual modificação na seção específica do código, alinhando-a às suas funcionalidades. O propósito primordial permanece na otimização do desempenho do processamento e na eficiência do retorno dos dados ao front-end.

3.3.2 Aplicação de Técnicas de Paralelização nos Gargalos Identificados

Com os gargalos identificados, a segunda etapa da metodologia concentra-se na aplicação de técnicas de paralelização direcionadas às áreas críticas previamente destacadas. O uso de primitivas de programação paralela, como threads ou processos, será considerado com base na natureza do problema. A ênfase estará na maximização da eficiência, garantindo que a paralelização ocorra de maneira consistente e sem introduzir potenciais problemas de concorrência.

3.3.3 Comparação do Desempenho Antes e Depois da Paralelização

A última etapa da metodologia consiste na avaliação comparativa do desempenho do código antes e depois da aplicação das técnicas de paralelização. Utilizando métricas apropriadas, será possível quantificar e qualificar os benefícios alcançados. A análise comparativa permite avaliar o impacto direto das alterações introduzidas no código, destacando ganhos significativos em termos de melhoria da aplicação.

Esta metodologia abrangente busca assegurar uma abordagem sistemática e eficaz na paralelização do código, promovendo melhorias substanciais em seu desempenho. Ao final deste processo, espera-se fornecer uma contribuição tangível para o avanço da eficiência computacional, alinhada com as demandas da plataforma em estudo.

3.4 Amostra e testes

A amostra de testes conduzida neste estudo consistiu em cinco séries de 10 repetições de exercícios para quatro grupos musculares, totalizando 50 repetições por grupo muscular

e 200 repetições no total. Esta amostra será utilizada em todas as análises realizadas neste trabalho, abrangendo desde a avaliação do comportamento da plataforma até a implementação de estratégias de paralelização.

Nesse contexto, empenhou-se na construção de uma base de dados coletados, visando estabelecer parâmetros para efetuar comparações adequadas e assegurar a fidedignidade na avaliação da eficiência das otimizações implementadas.

Para a estimativa dos tempos, torna-se crucial empregar valores medianos, a fim de desconsiderar possíveis tempos que estejam significativamente fora do padrão, devido a inconsistências temporárias.

4 CAPÍTULO 4

Neste capítulo, é apresentando a análise de comportamento da plataforma, assim como a perfilagem do código responsável pelo processamento dos dados da posição corporal do usuário.

4.1 Analisando o comportamento da plataforma

Para entender o funcionamento da plataforma em estudo, foram realizados testes de leitura de exercícios, a fim de verificar como está a comunicação da aplicação com o usuário. Os testes compreenderam a amostra citada na seção 3.4. Dentro das repetições de cada série, seis foram realizadas corretamente e quatro de forma propositalmente incorretas. Foram identificados três principais tipos de resultados, a saber:

- Erros mostrados: Quando o erro é reconhecido e impresso em tela;
- Erros coletados e não mostrados: Quando o erro é reconhecido, porém, por algum motivo, não foi impresso em tela;
- Erros não coletados: Quando algum erro não é reconhecido pela plataforma.

Grupo Muscular	Erros mostrados	Erros coletados e não mostrados	Erros não coletados
Braços	1	14	5
Pernas	4	15	1
Costas	4	11	5
Ombros	1	19	0

Tabela 1 – Leitura de exercícios com plataforma em formato originial

Fonte: Elaborado pelo autor.

Como pode ser visto na Tabela 1, de 80 erros que deveriam ser impressos em tela, apenas 10 tiveram o resultado esperado, um acerto de apenas 12,5%. Um ponto importante é que diversos movimentos errôneos foram calculados corretamente, porém, não foram impressos em tela, um total de 73,75%. Isso indica que o algoritmo consegue identificar que o usuário errou mas algum motivo faz com que esses feedbacks não sejam mostrados, isso foi percebido ao verificar os arquivos de log da plataforma, que se trata de uma base de dados de tudo que acontece dentro da aplicação. Além dessas observações, ainda houve uma taxa de 13,75% de erros não percebidos pela plataforma.

Diante do exposto, o estudo precisou caminhar para um melhor entendimento sobre o que está por trás dessa baixa taxa de sucesso. A partir daí, iniciou uma análise de como está o comportamento, desde a coleta de uma nova posição corporal via webcam do usuário até o momento em que um novo feedback é exibido em tela. Através da ferramenta nativa do navegador Google Chrome, o DevTools (ver figura 3), buscou-se:

- A quantidade de vezes, por segundo, em que uma nova posição corporal é enviada para processamento;
- Tempo total, desde o envio de nova posição corporal até o feedback ser impressa em tela;
- Proporção de tempo gasto, em relação a espera do retorno do dado tratado e ao tempo de renderização do feedback em tela.

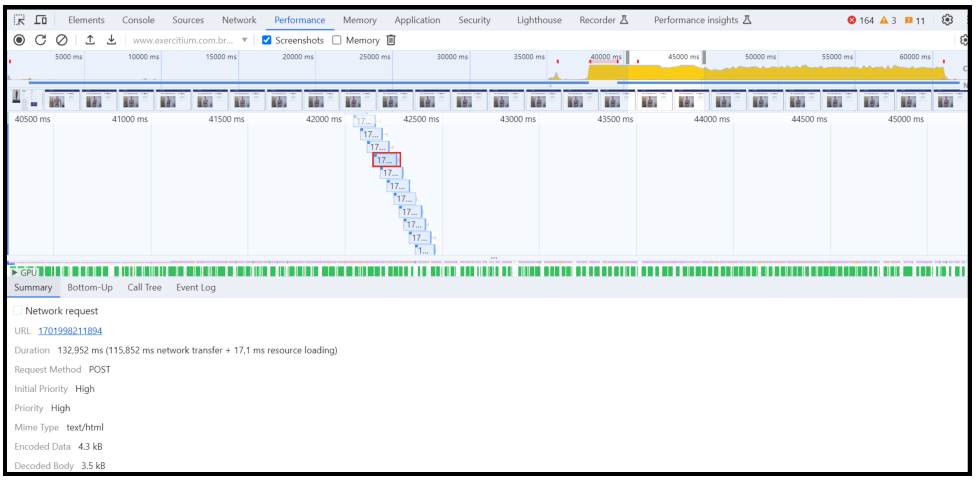

Figura 3 – *Print screen* do DevTools na aba de análise de performance.

Fonte: Elaborado pelo autor.

Nesta etapa, foram extraídos da amostra valores medianos em relação a tempo e frequência. Frequência essa, que teve seu valor obtido em 12 vezes por segundo. Isso significa a quantidade de vezes em um segundo em que, durante a leitura dos movimentos do usuário, é resgistrada e disparada uma nova posição para o back-end processar. Outro dado intrigante foi a frequência de resposta do dado processado dentro do mesmo período de um segundo, apenas 7,6 vezes. Essa diferença de quantidade de respostas e pedidos se dá justamente pelo tempo que o tratamento dos dados leva. Veja a equação 4.1.

$$\underbrace{f envio}_{\text{Frequência de envio}} > \underbrace{f resposta}_{\text{Frequência de resposta}} \tag{4.1}$$

A periodicidade da retroalimentação é determinada pelo intervalo em que um novo feedback é coletado para renderização em tela. Este procedimento é viabilizado pelo emprego

do método Javascript ‘window.requestAnimationFrame()‘, que notifica o navegador sobre a intenção de realizar uma animação, solicitando que o navegador invoque uma função específica para atualizar um quadro antes da próxima repintura (FOUNDATION, 2023). No contexto da aplicação em análise, a mencionada animação refere-se ao desenho e à inserção de informações no bloco de imagem proveniente da webcam do usuário. Na figura 4, pode ser visto o desenho realizado em tempo real em tela, representado por círculos nas articulações, além de informações de repetições, ângulos e tempos.

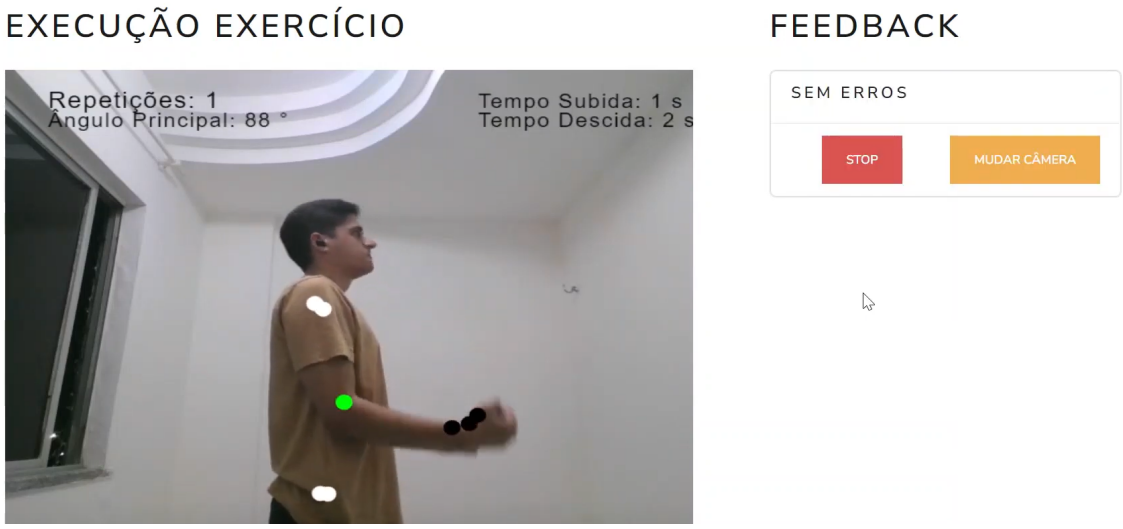

Figura 4 – *Print screen* da tela de execução de exercício na plataforma

Fonte: Elaborado pelo autor.

Adicionalmente, ao ser capturado por meio da mediana utilizando o DevTools, observou-se que a proporção de tempo despendido para a exibição de um novo feedback na tela, após o envio de uma nova posição para tratamento, é de aproximadamente 87% para o tempo de requisição (115,3 milissegundos) e 13% para o tempo de renderização na tela (16,5 milissegundos). Isso resulta em um período total de 132 milissegundos para a conclusão integral do ciclo de atualização de um feedback. Assim sendo, uma estratégia de otimização clara para essa ferramenta consistiria em acelerar esse ciclo, buscando equiparar, ou ao menos aproximar, as frequências de envio e recebimento.

4.2 Realizando a perfilagem

Agora visto que o tempo em que o ciclo de impressão de novos feedbacks em tela precisa ser acelerado, buscou-se averiguar oportunidades de otimizações dentro do código responsável pelo processamento dos dados. Para isso, foi realizada a perfilagem do código das etapas 4 e 5, como mostrado na figura Figura 1, utilizando as bibliotecas cProfile e pstats, para coletar suas estatísticas. O algoritmo 4.1, mostra como parte esse processo foi realizado.

Algoritmo 4.1 – Código para perfilagem

```
import cProfile
import pstats

def funcao_ativa_backend():
    pr.enable()
    # ... Codigo a ser analisado ...
    pr.disable()

    # Carrega estatisticas se o arquivo ja existir
    try:
        existing_stats = pstats.Stats('profile_result')
        existing_stats.add(pr)
        pr = existing_stats
    except FileNotFoundError:
        pass  # O arquivo ainda nao existe

    # Salva as estatisticas no arquivo
    with open('profile_result', 'wb') as f:
        pr.dump_stats(f.name)

if __name__ == "__main__":
    pr = cProfile.Profile()
```

Fonte: Elaborado pelo autor.

A estrutura lógica do código de perfilagem reside em iniciar a coleta de dados sempre que a função que aciona o back-end for invocada. Ao concluir a execução, a coleta de dados é encerrada. Buscando entender cada etapa do algoritmo 4.1, uma breve explicação pode ser vista abaixo:

1. import cProfile, pstats: Importa os módulos necessários
2. pr = cProfile.Profile(): Cria uma instância para coleta de estatísticas
3. pr.enable(): Ativa a perfilagem
4. pr.disable(): Desativa a perfilagem
5. Bloco 'try': Verifica-se a existência de arquivo de armazenamento
6. with open('profile_result...: Arquivo de armazenamento será criado ou sobrescrito
7. pr.dump_stats(f.name): Método para salvar as estatísticas no arquivo

Antes de analisar o resultado da perfilagem, se faz necessário entender a arquitetura do código que é responsável por todo esse processamento de dados. Veja a figura 5 a seguir:

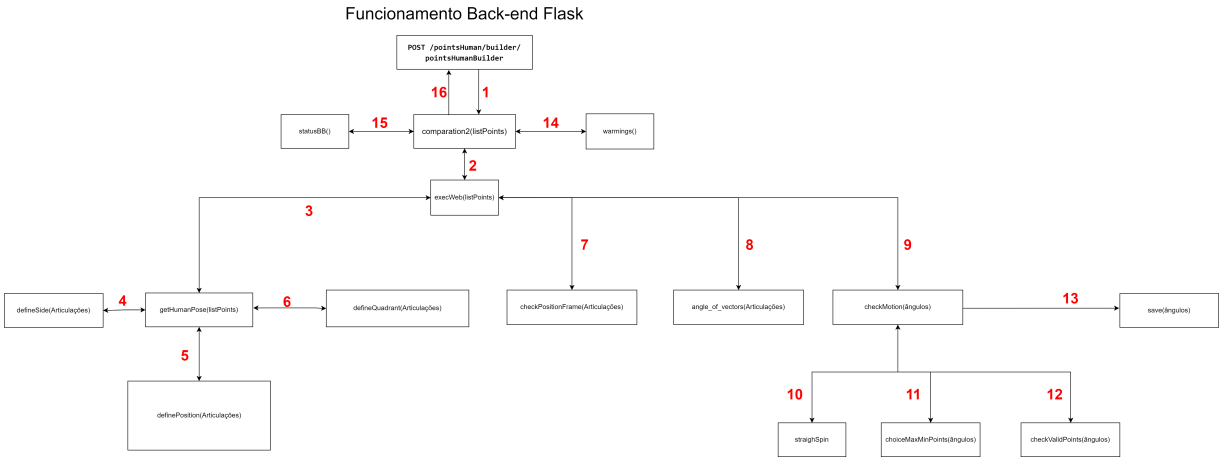

Figura 5 – *Print screen* da arquitetura do backend de processamento.

Fonte: Elaborado pelo autor.

É possível ver que existem 16 etapas, desde o momento em que o código recebe uma nova requisição até o envio da resposta tratada. Cada etapa, enumerada de acordo com a figura 5, pode ser entendida nos tópicos abaixo:

1. Back-end recebe uma nova solicitação de tratamento. No corpo da requisição há uma lista de pontos que significam a posição atual do usuário, a função comparation2 é chamada;
2. A subfunção execWeb é acionada, recebendo a lista de pontos;
3. Função responsável por entender a lista de pontos é chamada;
4. A função defineSide() entende de qual lado está o usuário, virado para esquerda ou direita;
5. Nesta etapa, é calculado o que cada ponto da lista representa nos eixos X,Y e Z;
6. É definido em qual quadrante da imagem cada articulação está posicionada;
7. Verifica se alguma articulação está fora do enquadramento da imagem;
8. Calcula-se o ângulo atual, para cada articulação, da posição do usuário;
9. Função checkMotion é chamada para verificar correções de ângulos;
10. É verificado se a angulação de inclinação do pescoço e coluna está de acordo com a do profissional;

11. Os ângulos calculados na etapa 8 são verificados se estão fora da margem de erro do profissional;
12. Verifica quantos graus, de cada angulação fora da margem de erro, precisa ser movida para voltar a posição correta;
13. Os novos dados tratados são salvos para servir na geração de relatório;
14. O texto do feedback de correção é montado, caso exista;
15. É montado o JSON de retorno com os dados atualizados, incluindo feedback, número de repetições, angulações, etc...;
16. Resposta tratada é enviada ao front-end.

A condução do processo de perfilagem abarcou desde a etapa 1 até a 16. O objetivo primordial consistiu em identificar quais funções poderiam representar um gargalo para o ciclo completo de atualização de feedbacks. Subsequentemente a essa análise, foi gerado um arquivo, referenciado no algoritmo 4.1, denominado 'profile_results'. Mediante a utilização da ferramenta Snakeviz, tornou-se possível realizar uma análise visual do desempenho de cada etapa durante os testes. Consulte a figura 6 para obter mais detalhes:

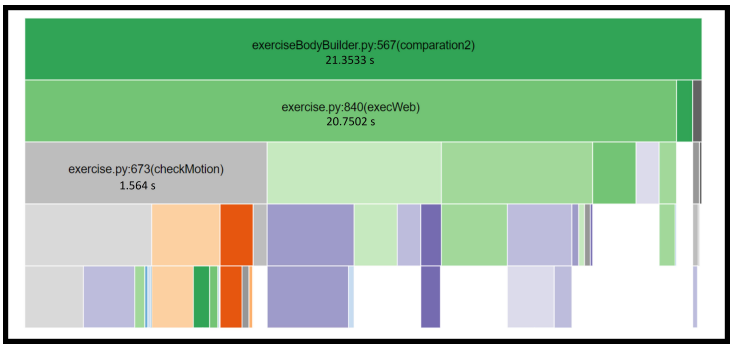

Figura 6 – *Print screen* da tela de resultado da perfilagem do Snakeviz.

Fonte: Elaborado pelo autor.

O comprimento de cada barra denota a duração durante a qual a função permaneceu ativa em relação ao tempo total de execução do back-end. Dentro de cada barra, encontramos três valores significativos, os quais são formatados da seguinte maneira: nome do arquivo; (Nome da função); (Tempo acumulado dessa função e suas subfunções). Entretanto, para obter uma fundamentação mais aprofundada, são necessários outros dados, como o tempo de execução da própria função, a quantidade de execuções, entre outros. A ferramenta Snakeviz também oferece a capacidade de visualizar uma tabela contendo esses dados, veja a Tabela 2 a seguir:

ncalls	**tottime/ncalls**	**cumtime/ncalls**	**function**
208	0.004178	0.1027880	comparation2
208	0.01122	0.099649442	execWeb
208	0.007525	0.05358118	statusBB
208	0.003611	0.03722410	checkMotion
208	0.01116	0.026277921	getHumanPose
626	0.005686	0.00761371	angle_of_vectors
208	0.01515	0.019431064	choiceMaxMinPoints
626	0.007008	0.00437401	definePositon
208	0.01076	0.0099789039	straightSpin
208	0.005665	0.0050723881	checkValidPoints
626	0.00203	0.001184761117	defineQuadrant
626	0.001987	0.0011550069	checkPositionFrame
626	0.001704	0.0010245528	defineSide
208	0.004159	0.0026200752	save
8	0.01546	0.037397141834	warnings
8	0.008395	0.02716497843	defineErros

Tabela 2 – Resultado da perfilagem

Fonte: Elaborado pelo autor.

É importante lembrar que os valores da tabela 2 são baseados na mediana das medidas feitas para as todas as cinco séries de 10 repetições para cada grupo muscular. A coluna 'ncalls' informa o número de vezes que a função foi chamada, 'tottime/ncalls' é o tempo gasto na função, sem contar subfunções, por cada chamada, e 'cumtime/ncalls' indica o tempo total gasto na função e subfunções dividido pela quantidade de chamadas. A ordenação mostrada está em formato decrescente de 'cumtime', que é o tempo total gasto na função e suas subfunções, assim como na figura 6.

5 CAPÍTULO 5

Neste capítulo, são abordados técnicas de paralelização de código e resultados comparativos pós-otimizações.

5.1 Escolhendo onde paralelizar

Identificar qual parte do código paralelizar pode não ser uma tarefa tão simples. Como já discutido no Capítulo 2, a condição de corrida e compartilhamento de variáveis precisam ser levados em consideração junto também com toda a análise de perfilagem. O primeiro passo é perceber, de acordo com a tabela 2, quais funções valem mais a pena investir tempo. As colunas 'ncalls' e 'tottime/ncalls' demonstram um certo grau de criticidade, visto que esse é o tempo gasto diretamente na função citada, sem contar suas subfunções. Esse é um dos indicativos de possíveis gargalos. Seguindo a ordenação das 5 funções com maior 'tottime/ncalls':

1. warnings;
2. choiceMaxMinPoints;
3. execWeb;
4. getHumanPose;
5. straightSpin;

Porém, deve-se salientar que, por mais que a função warnings seja a que mais gasta tempo por chamada, ela só foi ativada 8 vezes, enquanto diversas outras tiveram mais de 200 ativações. Isso pode fazer com que o ganho final não seja tão relevanta ao paralelizar esse trecho do código, pois é executado apenas em casos de erros coletados. Portanto, para gerar um melhor embasamento para tomada de decisão, vejamos a ordenação de acordo com o 'tottime', sem levar em consideração a quantidade de chamadas 'ncalls'. Veja abaixo:

1. definePositon;
2. angle_of_vectors;
3. choiceMaxMinPoints;
4. execWeb;

5. getHumanPose;

Percebe-se que temos três funções que aparecem dentro duas ordenações citadas, são elas: 'choiceMaxMinPoints', 'execWeb' e 'getHumanPose'. São funções que acabam tendo uma evidência um pouco maior de possível gargalo devido ao tempo que elas gastam no total e por ativação. O próximo passo se trata de aprofundar-se nesses trechos do código com o objetivo de entender se são de fato paralelizáveis ou podem gerar problemas de resultado caso mais de um processador trabalhe ao mesmo tempo. O código em 'choiceMaxMinPoints' é voltado para calcular quais os ângulos que estão fora de uma margem de erro do movimento previamente cadastrado pelo profissional, ou seja, verifica se o ângulo está acima ou abaixo de uma variável máxima e mínima. Dentro desse processo, existe um momento onde calcula-se a amplitude do movimento em três dimensões, determina qual dimensão tem a maior amplitude absoluta e, em seguida, atribui o valor correspondente a essa dimensão à uma variável. Veja o trecho na figura 7 abaixo:

```
elif self.statusMotion == "Up":
    range = [
        self.anglesDown[0][-1] - self.initial[0],
        self.anglesDown[1][-1] - self.initial[1],
        self.anglesDown[2][-1] - self.initial[2]
        ]
    res = [abs(ele) for ele in range]
    self.index = res.index(max(res))
    self.mainAngle = angle[self.index]

elif self.statusMotion == "Down":
    range = [
        self.anglesUp[0][-1] - self.initial[0],
        self.anglesUp[1][-1] - self.initial[1],
        self.anglesUp[2][-1] - self.initial[2]
        ]
    res = [abs(ele) for ele in range]
    self.index = res.index(max(res))
    self.mainAngle = angle[self.index]
```

Figura 7 – *Print screen* de trecho de código da função choiceMaxMinPoints.

Fonte: Elaborado pelo autor.

As condicionais existentes no código, representadas por 'elif self.statusMotion == "Up"' e 'elif self.statusMotion == "Down"' verificam se o movimento atual do usuário está em descida ou subida. Logo após, 'range' é uma lista que armazena a amplitude em cada uma das três dimensões. Essa amplitude é calculada subtraindo os valores iniciais (self.initial), que são buscados da posição anterior do usuário, dos valores finais (self.anglesDown ou self.anglesUp) em cada dimensão. Portanto, 'range' é uma lista com três elementos, representando a diferença entre os valores finais e iniciais nas três dimensões. Outra lista, chamada 'res', armazena os valores absolutos de cada elemento em 'range' através de

uma estrutura de repetição representada por '[abs(ele) for ele in range]'. Posteriormente, 'self.index' é atribuído ao índice do valor máximo em 'res'. Isso significa que 'self.index' indicará qual das três dimensões tem a maior amplitude em termos absolutos. Por último, 'self.mainAngle' é atribuído ao valor correspondente ao índice determinado em 'self.index' na lista 'angle', lista essa que é recebida como argumento da função. Em resumo, esse trecho diz qual a posição, dentre as três dimensões, está com a maior amplitude, onde posteriormente é verificado se essa amplitude implica em uma posição errada ou não.

Dentro desta função, esse trecho é o mais promissor para aplicar a paralelização, visto que o restante do código trabalha apenas com condicionais e trechos que necessitam ser sequenciais devido ao compartilhamento de variáveis globais. Perceba que, dentro da estrutura de repetição da figura 7, não há qualquer dependência de valores por trabalhar apenas com atribuição de valores que serão lidos posteriormente na função. Portanto, demonstra ser um trecho possível de utilizar mais de um processador, apesar de não necessariamente ser o trecho da função que realize mais trabalho.

Seguindo esse modelo, analisou-se a função 'execweb', no qual pode ser vista na figura 8 abaixo:

```
if len(lmList) != 0 and len(lmList[11]) != 0 and len(lmList[12]) != 0:

    self.poseHuman.getHumanPose(lmList, self.changeSide)
    n = self.poseHuman.numberArticulation

    for idArticulation in range (n):
        framed = self.poseHuman.checkPositionFrame(idArticulation, width, height)
        self.angle_of_vectors([], idArticulation, False)

    (angleDown, angleUp, lastedUp, lastedDown) = self.checkMotion(self.angle)
    self.save(angleDown, angleUp, lastedUp, lastedDown)

    self.indexArt = self.poseHuman.indexArt

return  angleDown, angleUp, framed
```

Figura 8 – *Print screen* de trecho de código da função execWeb.

Fonte: Elaborado pelo autor.

Nesta etapa, destaca-se alguns pontos. A condicional, representada pela primeira linha, apenas verifica se a lista de articulações não estão vazias. Após isso, a função 'getHumanPose' é chamada, no qual é uma das funções que aparecem com mais tempo gasto ao levar em consideração os critérios de 'tottime/ncalls' e 'tottime' coletados através da tabela 2. Posteriormente, existe uma estrutura de repetição, do tamanho da quantidade de articulações que estão sendo lidas no exercício, onde são chamadas as funções 'checkPositionFrame' e 'angle_of_vectors'. Essa estrutura de repetição é responsável por verificar se as articulações estão dentro do enquadramento da câmera (checkPositionFrame) e calcular e armazenar o ângulo entre dois vetores em um espaço bidimensional (angle_of_vectors).

Ao realizar tal verificação, a função 'execWeb' tende a ser uma boa oportunidade de paralelização se suas subfunções de dentro da estrutura de repetição permitirem que

diferentes processos trabalhem ao mesmo tempo entre si. A função checkPositionFrame se apresenta pelo código apresentado na figura 9 a seguir:

```
if (
    self.movementAxisA[idArticulation][0] > self.width or
    self.movementAxisB[idArticulation][0] > self.width or
    self.stoppedAxisA[idArticulation][0]  > self.width or
    self.stoppedAxisB[idArticulation][0] > self.width or
    self.movementAxisA[idArticulation][1] > self.height or
    self.movementAxisB[idArticulation][1] > self.height or
    self.stoppedAxisA[idArticulation][1]  > self.height or
    self.stoppedAxisB[idArticulation][1] > self.height
    ):

    timesWrong += 1

else:
    timesRight += 1

if timesWrong > 0:
    timesRight = 0
    timesWrong = 0
    return False
elif timesRight > 0:
    timesRight = 0
    timesWrong = 0

return True
```

Figura 9 – *Print screen* de trecho de código da função checkPositionFrame.

Fonte: Elaborado pelo autor.

Essa função recebe todas as posições de articulações em uma visão bidimensional e verifica se elas estão enquadradas na imagem da câmera do usuário através das variáveis de lagura e altura representadas por 'width' e 'height'. Ao seu fim, é retornado um valor booleano (verdadeiro ou falso). Esta etapa aparenta ser paralelizável por dois motivos, o primeiro é que os valores das variáveis 'width' e 'height' são fixos, sem necessitar de atualização de outra fonte, o segundo é que as atribuições feitas nas variáveis 'timesWrong' e 'timesRight' são locais e a cada execução elas são zeradas.

Já na função 'angle_of_vectors', temos o código da figura 10.

```
def angle_of_vectors(self, img = [], idArticulation = 0):

    lineYA = self.poseHuman.movementAxisB[idArticulation][1]- self.poseHuman.movementAxisA[idArticulation][1]
    lineXA = self.poseHuman.movementAxisA[idArticulation][0] - self.poseHuman.movementAxisB[idArticulation][0]
    lineYB = self.poseHuman.stoppedAxisA[idArticulation][1]- self.poseHuman.stoppedAxisB[idArticulation][1]
    lineXB = self.poseHuman.stoppedAxisB[idArticulation][0]- self.poseHuman.stoppedAxisA[idArticulation][0]

    dotProduct0 = lineXA*lineXB + lineYA*lineYB

    modOfVector0 = math.sqrt( lineXA*lineXA + lineYA*lineYA)*math.sqrt(lineXB*lineXB + lineYB*lineYB)

    angle0 = dotProduct0/modOfVector0

    if self.poseHuman.stoppedAxis[idArticulation] == "ombro-quadril" and self.poseHuman.movementAxis[idArticulation] == "ombro-cotovelo":
        angleInDegree0 = 180 - math.degrees(math.acos(angle0))

    else:
        angleInDegree0 = math.degrees(math.acos(angle0))

    self.angle[idArticulation] = round(angleInDegree0)
```

Figura 10 – *Print screen* de trecho de código da função angle_of_vectors.

Fonte: Elaborado pelo autor.

Essa etapa é responsável por calcular o ângulo entre dois vetores em um espaço bidimensional. Nas primeiras linhas, as variáveis 'lineYA', 'lineXA', 'lineYB' e 'lineXB' recebem o vetor resultante subtraído das coordenadas X ou Y do ponto final (A ou B) pelas

coordenadas X ou Y do ponto inicial (A ou B) para o eixo de movimento. Posteriormente é realizado o produto escalar entre os dois vetores, o produto dos módulos (normas) dos dois vetores, o cosseno do ângulo entre os dois vetores usando o produto escalar e as norma, por fim, o ângulo é calculado e armazenado na lista 'self.angle' na posição 'idArticulation'. Portanto, é percebido que a função é independente em cada iteração, já que sua chamada opera em uma articulação diferente (idArticulation) por vez, o que indica que não há dependências.

Por fim, a última função analisada foi a 'getHumanPose'. Veja figura 11:

```
def getHumanPose(self, lmList):

    self.lmList = lmList

    for idArticulation in range(self.numberArticulation):

        self.definePositon(idArticulation)
        self.defineQuadrant(idArticulation)
```

Figura 11 – *Print screen* de trecho de código da função getHumanPose.

Fonte: Elaborado pelo autor.

Novamente é visto uma estrutura de repetição do tamanho do número de articulações, dentro do bloco são chamadas duas funções, que são 'definePositon' e 'defineQuadrant'. A primeira delas, tem o objetivo de atribuir os valores da lista de pontos, recebidas pelo fron-end, para alguma articulação. Já na segunda função, é tratado à lógica de determinar o quadrante de um ponto no plano com base em comparações entre as coordenadas X e Y. As duas funções podem ser vistas nas figuras 12 e 13 a seguir:

```
def definePositon(self, idArticulation):

    if (self.stoppedAxis[idArticulation] == "ombro-cotovelo"):

        if self.side[idArticulation] == "Left":

            if len(self.lmList[11]) != 0 :
                self.stoppedAxisA[idArticulation] = [self.lmList[11][1], self.lmList[11][2]]
                self.ligationAxisA[idArticulation] = [self.lmList[11][1], self.lmList[11][2]]
                if len(self.lmList[11+1]) != 0 :
                    self.ligationAxisB[idArticulation] = [self.lmList[14][1], self.lmList[14][2]]

            if len(self.lmList[13]) != 0 :
                self.stoppedAxisB[idArticulation] = [self.lmList[13][1], self.lmList[13][2]]

            if idArticulation == 0 : self.indexArt = 13

        else:

            if len(self.lmList[11+1]) != 0  :
                self.stoppedAxisA[idArticulation] = [self.lmList[11+1][1], self.lmList[11+1][2]]
                self.ligationAxisA[idArticulation] = [self.lmList[11+1][1], self.lmList[11+1][2]]
                if len(self.lmList[11]) != 0 :
                    self.ligationAxisB[idArticulation] = [self.lmList[13][1], self.lmList[13][2]]

            if len(self.lmList[13+1]) != 0  :
                self.stoppedAxisB[idArticulation] = [self.lmList[13+1][1], self.lmList[13+1][2]]

            if idArticulation == 0 : self.indexArt = 14
```

Figura 12 – *Print screen* de trecho de código da função definePosition.

Fonte: Elaborado pelo autor.

```
def defineQuadrant(self, idArticulation):

    if (self.center[idArticulation][0] < self.movementAxisValue[idArticulation][0]):

        if (self.side[idArticulation] == "Left"):
            self.goBack[idArticulation] = False
            self.goFront[idArticulation] = True
        else:
            self.goBack[idArticulation] = True
            self.goFront[idArticulation] = False

        if(self.center[idArticulation][1] >=  self.movementAxisValue[idArticulation][1]):
            self.QuadrantM[idArticulation] = 1

        else:
            self.QuadrantM[idArticulation] = 4

    else:
        if (self.side[idArticulation] == "Left"):
            self.goBack[idArticulation] = True
            self.goFront[idArticulation] = False
        else:
            self.goBack[idArticulation] = False
            self.goFront[idArticulation] = True

        if(self.center[idArticulation][1] <=  self.movementAxisValue[idArticulation][1]):
            self.QuadrantM[idArticulation] = 3
        else:
            self.QuadrantM[idArticulation] = 2
```

Figura 13 – *Print screen* de trecho de código da função defineQuadrant.

Fonte: Elaborado pelo autor.

As duas subfunções de 'getHumanPose' são independentes por se tratarem de articulações diferentes a cada iteração, assim como ocorre em funções anteriormente discutidas. Resultando assim, em três funções que foram escolhidas para paralelizar, com o objetivo de diminuir o tempo total do ciclo de correções corporais da plataforma. As funções são:

1. choiceMaxMinPoints;
2. execWeb;
3. getHumanPose;

5.2 Utilizando OpenMP em Python

O OpenMP, como já discutido no capítulo 2 deste trabalho, é amplamente utilizado para programação paralela em linguagens como C, C++, e Fortran. No entanto, o Python, que é a linguagem do back-end de processamento, não oferece suporte nativo para o OpenMP. Para contornar essa limitação, foi adotada uma abordagem híbrida, integrando o OpenMP com Python. A solução envolve a criação de um arquivo em C que contém o código OpenMP desejado e, em seguida, importá-lo no código Python, estabelecendo uma comunicação eficiente entre as duas linguagens (DIGITALOCEAN, 2023). A seguir, são descritos os passos necessários para a implementação dessa abordagem:

1. Criação do código em C que utiliza o OpenMP para paralelizar as partes relevantes do algoritmo;

2. Utilização de um compilador C compatível com o OpenMP para compilar o código em um arquivo compartilhado;

3. Escrita de uma interface Python utilizando a biblioteca ‘ctypes‘ para carregar a biblioteca compartilhada gerada na etapa anterior;

4. Declaração e chamada das funções em Python correspondentes às funções definidas no código C;

Essa abordagem híbrida permite que seja utilizado o OpenMP em conjunto com Python, tirando vantagem da eficiência de execução paralela oferecida pela API.

5.2.1 OpenMP na função choiceMaxMinPoints

Seguindo os passos descritos anteriormente, a função ’choiceMaxMinPoints’ foi paralelizada. Na figura 14 encontra-se o código em C que representa a estrutura de repetição que será chamado no Python.

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

void calcular_abs_max(int size, double *range, int *index) {
    int i;
    double *res = (double *)malloc(size * sizeof(double));

    #pragma omp parallel for
    for (i = 0; i < size; i++) {
        res[i] = fabs(range[i]);
    }

    *index = 0;
    for (i = 1; i < size; i++) {
        if (res[i] > res[*index]) {
            *index = i;
        }
    }

    free(res);
}
```

Figura 14 – *Print screen* de trecho de código da função choiceMaxMinPoints escrita em C utilizando OpenMP.

Fonte: Elaborado pelo autor.

A função calcular_abs_max recebe um tamanho (size) e um ponteiro para um array de números (range). Utiliza OpenMP para calcular o valor absoluto de cada elemento em paralelo. Encontra o índice do maior valor absoluto no array. O próximo passo foi compilar o código C utilizando o GCC, ver prompt 5.1. A flag -fopenmp habilita o suporte a OpenMP. O resultado é um arquivo compartilhado (choiceMaxMinPoints.so) que contém a função paralelizada.

$$gcc - fopenmp - ochoiceMaxMinPoints.so - shared - fPICchoiceMaxMinPoints.c \tag{5.1}$$

```python
import ctypes

# Carregar a biblioteca compartilhada
choiceMaxMinPointsC_lib = ctypes.CDLL('./choiceMaxMinPoints.so')

# Definir a assinatura da função em C
choiceMaxMinPointsC_lib.calcular_abs_max.restype = None
choiceMaxMinPointsC_lib.calcular_abs_max.argtypes = [
    ctypes.c_int,      # Tamanho da lista 'range'
    ctypes.POINTER(ctypes.c_double),  # Ponteiro para a lista 'range'
    ctypes.POINTER(ctypes.c_int),     # Ponteiro para o índice resultante
]

def calcular_abs_max_paralelo(range):
    size = len(range)
    range_c = (ctypes.c_double * size)(*range)
    index_c = ctypes.c_int()

    # Chamar a função em C
    choiceMaxMinPointsC_lib.calcular_abs_max(size, range_c, ctypes.byref(index_c))

    # Retornar os resultados
    return index_c.value
```

Figura 15 – *Print screen* de trecho de código da função choiceMaxMinPoints escrita em Python chamando o arquivo em c.

Fonte: Elaborado pelo autor.

Na figura 15 acima, a biblioteca ctypes foi usada para carregar a biblioteca compartilhada (choiceMaxMinPoints.so). Definiu-se a assinatura da função em C para que Python possa chamá-la e informou ao algoritmo qual seria o tipo de dado que estaria sendo recebido. Abaixo, na figura 16, é possível ver a nova função já sendo chamada no código original da plataforma. Perceba que agora já não é mais necessário utilizar a estrutura de repetição antes vista na figura 7, pois agora essa estrutura está dentro do código em C, que está sendo chamado no código original.

```python
elif self.statusMotion == "Up":
    range = [
        self.anglesDown[0][-1] - self.initial[0],
        self.anglesDown[1][-1] - self.initial[1],
        self.anglesDown[2][-1] - self.initial[2]
        ]

    self.index = self.calcular_abs_max_paralelo(range)
    self.mainAngle = angle[self.mostIndex]

elif self.statusMotion == "Down":
    range = [
        self.anglesUp[0][-1] - self.initial[0],
        self.anglesUp[1][-1] - self.initial[1],
        self.anglesUp[2][-1] - self.initial[2]
        ]

    self.index = self.calcular_abs_max_paralelo(range)
    self.mainAngle = angle[self.mostIndex]
```

Figura 16 – *Print screen* de trecho de código da função choiceMaxMinPoints chamando a função feita em C.

Fonte: Elaborado pelo autor.

5.2.2 OpenMP na função execWeb

Na função 'execWeb', houve uma mudança no processo de paralelização. Ao invés de escrever todo o código que está dentro de suas subfunções na liguagem C, optou-se por criar um módulo, com API Python/C, que permite a chamada de funções Python diretamente em um código C (PRAKASH, 2019). As funções angle_of_vectors() e checkPositionFrame(), das figuras 10 e 9, foram inseridas em um arquivo chamado 'moduleExecWeb.py'. Após isso, um arquivo chamado execWebC.c foi criando contendo seu código, veja figura 17 abaixo.

```c
#include <stdio.h>
#include <omp.h>

void checkPositionFrame(int idArticulation, int width, int height);
void angle_of_vectors(int idArticulation);

void processar_articulacoes(int n, int width, int height) {
    #pragma omp parallel for
    for (int idArticulation = 0; idArticulation < n; idArticulation++) {
        checkPositionFrame(idArticulation, width, height);
        angle_of_vectors(idArticulation);
    }
}
```

Figura 17 – *Print screen* de trecho de código da parte paralelizável da função execWeb c.

Fonte: Elaborado pelo autor.

Na próxima etapa, o arquivo 'execWebParallel.py' foi criado, como pode ser visto na figura 18.

```python
from ctypes import CDLL, c_int

execWebC_lib = CDLL('./execWebC.so')

execWebC_lib.checkPositionFrame.argtypes = [c_int, c_int, c_int]
execWebC_lib.angle_of_vectors.argtypes = [c_int]

def checkPositionFrame(idArticulation, width, height):
    execWebC_lib.checkPositionFrame(idArticulation, width, height)

def angle_of_vectors(idArticulation):
    execWebC_lib.angle_of_vectors(idArticulation)
```

Figura 18 – *Print screen* do código do arquivo compartilhado que será chamado.

Fonte: Elaborado pelo autor.

Por último, bastou compilar o código execWebC.c contendo OpenMP, com o prompt 5.2, e módulo Python/C, com o prompt 5.3.

$$gcc - fopenmp - oexecWebC.so - shared - fPICexecWebC.c \tag{5.2}$$

$$gcc - oexecWebParallel.so - shared - fPIC \\ - I/usrVini/include/python3.10 \\ execWebParallel.c - lpython3.10 \quad (5.3)$$

O código original da função 'execWeb', com a chamada atualizada, pode ser visto na figura 19. Perceba que, como a estrutura de repetição já está contida no arquivo em C, não foi necessário mantê-la no código original.

```
if len(lmList) != 0 and len(lmList[11]) != 0 and len(lmList[12]) != 0:

    self.poseHuman.getHumanPose(lmList, self.changeSide)

    n = self.poseHuman.numberArticulation
    framed = self.checkPositionFrame(n, width, height)
    self.angle_of_vectors([], n, False)
```

Figura 19 – *Print screen* de trecho do código paralelizado da função 'execWeb'.

Fonte: Elaborado pelo autor.

5.2.3 OpenMP na função getHumanPose

A função 'getHumanPose', vista na figura 11, teve o mesmo processo de paraelização da 'execWeb', no sentido de criar módulos em Python para chamar em C e depois inserir no código original. Foi criado o módulo 'getHumanPose.py'. Em seguida, foi escrita uma interface C/Python no arquivo 'getHumanPoseInterface.c'. O código C foi então compilado com suporte ao OpenMP utilizando o GCC. Com a biblioteca compartilhada gerada, o arquivo de interface Python usando a biblioteca 'ctypes' foi criado.

Assim, o trecho do código original, com a chamada para o arquivo em C com OpenMP, foi atualizado e pode ser visto na figura 20. A estrutura de repetição também está inserida no arquivo em C, logo, não se faz mais necessário no código original.

```
def getHumanPose(self, lmList):

    self.lmList = lmList

    self.definePositon(self.numberArticulation)
    self.defineQuadrant(self.numberArticulation)
```

Figura 20 – *Print screen* de trecho do código paralelizado da função 'getHumanPose'.

Fonte: Elaborado pelo autor.

5.3 Resultados obtidos

Os resultados obtidos após a aplicação das paralelizações de código no âmbito deste trabalho revelam uma breve melhoria no ciclo de coleta de feedback. A plataforma foi testada novamente com 200 repetições de exercícios, utilizando a mesma amostra que foi previamente empregada antes da intervenção. Veja a Tabela 3 com os resultados.

Grupo Muscular	Erros mostrados	Erros coletados e não mostrados	Erros não coletados
Braços	4	14	2
Pernas	7	11	2
Costas	5	10	5
Ombros	6	13	1

Tabela 3 – Leitura de exercícios com plataforma em formato paralelizado

Fonte: Elaborado pelo autor.

Apesar de ainda demonstrar a necessidade de uma longa melhoria pela frente, em comparação com coleta pré-otimizações, houve uma melhora. Dos 10 erros que foram impressos em tela pré-otimização, de um total de 80, dessa vez foram impressos 22. Em outras palavras, uma melhora de 120% nos resultados impressos. Os feedbacks de correção que foram calculados corretamente, porém não foram impressos em tela, caíram de 73,75% dos casos para 60%. Os erros não coletados também diminuíram, de 13,75% para 12,5%. Em contrapartida, dois grupos musculares tiveram o aumento de uma unidade de erros não percebidos pela aplicação, no quais foram pernas e ombros.

Do mesmo modo em que foi analisado a comunicação da plataforma com o usuário, com o auxílio da ferramenta DevTools, os mesmos testes foram realizados pós-paralelização com o objetivo de verificar se houve melhora no tempo de ciclo total de coleta de feedback. A frequência de requisições, em que o front-end envia uma posição corporal para o back-end, teve um leve aumento de 12 vezes por segundo para 15 vezes por segundo. A frequência de resposta do dado processado dentro do mesmo período, de um segundo, também aumentou, saiu de 7,6 vezes para 11 vezes. Em outras palavras, a plataforma ainda está solicitando ao back-end um tratamento de resposta mais rápido, porém essa diferença diminuiu. A equação 4.1 ainda representa como está a situação atual, entretanto, cada vez mais próximo de igualar a frequência de coleta e de resposta.

Também se observou, que a proporção de tempo despendido para a exibição de um novo feedback na tela diminuiu. Saindo de um total de 132 milissegundos para 117 milissegundos, onde aproximadamente 86% para o tempo de requisição (100,1 milissegundos) e 14% para o tempo de renderização na tela (16,4 milissegundos). Os 15 milissegundos de melhora no tempo geral foram devido ao desempenho otimizado, causado pela paralelização das funções apresentadas.

Por fim, foi realizada a perfilagem do novo código utilizando as mesmas ferramentas, metodologia e amostra de testes. O resultado pode ser visto na tabela 4 a seguir.

ncalls	**tottime/ncalls**	**cumtime/ncalls**	**function**
252	0.00359428	0.07901236595	comparation2
252	0.0096612	0.07571363022	execWeb
252	0.007525	0.05358118	statusBB
252	0.00310746	0.02901933231	checkMotion
252	0.0095864	0.02048699327	getHumanPose
723	0.00489096	0.00595155343	angle_of_vectors
252	0.013029	0.01513405149	choiceMaxMinPoints
723	0.00604488	0.00341044831	definePositon
252	0.01076	0.0099789039	straightSpin
252	0.005665	0.0050723881	checkValidPoints
723	0.001748	0.00092453728	defineQuadrant
723	0.00171182	0.00090000483	checkPositionFrame
723	0.001704	0.0010245528	defineSide
252	0.004159	0.0026200752	save
9	0.01546	0.037397141834	warnings
9	0.008395	0.02716497843	defineErros

Tabela 4 – Resultado da perfilagem pós-paralelização

Fonte: Elaborada pelo autor.

6 CONCLUSÕES, LIMITAÇÕES E RECOMENDAÇÕES

6.0.1 Conclusões

Ao término desta investigação, emergiram conclusões cruciais acerca dos objetivos propostos no início deste estudo. Inicialmente, conduziu-se uma análise do comportamento da plataforma, considerando todo o ciclo de correção de posição corporal. Uma das métricas primordiais revelou que a plataforma efetua requisições de maneira mais ágil do que a aplicação consegue processar, enquanto um desafio adicional se evidenciou na quantidade de erros propositais exibidos na tela durante os testes, demandando uma taxa de sucesso mais elevada. Muitos desses erros decorrem do tempo de coleta de feedback, que deve acompanhar o movimento do usuário em alta velocidade.

No tocante ao objetivo específico de analisar gargalos de desempenho da plataforma, a coleta de dados proveniente da análise do comportamento da aplicação e da perfilagem proporcionou insights significativos para melhorias. Notou-se que mais de 85% do tempo entre as requisições de leitura da posição corporal do usuário é consumido no tratamento dos dados. Embora o processamento integral dos dados não seja trivial, a perfilagem revelou gargalos que impediam um melhor desempenho da aplicação. Algumas funções foram identificadas como oportunidades para paralelização devido ao tempo em que o processo levava para sair desses passos, destacando-se como alvos para otimização.

Posteriormente, procedeu-se à paralelização de três funções, empregando a API OpenMP em conjunto com técnicas para lidar com as linguagens Python e C. Este processo mostrou-se promissor e relevante, resultando em um aumento de 120% na taxa de sucesso de impressão de erros em tela em uma máquina com 2 processadores e que roda em nuvem. Entretanto, persiste um número considerável de feedbacks que deveriam ser exibidos na tela, mas não o são, representando ainda mais de 70% das ocorrências, embora anteriormente à paralelização essa proporção alcançasse quase 90%. Vale ressaltar que a plataforma consegue coletar mais de 85% de todos os erros dos usuários, porém ainda se mantém com uma grande dificuldade em manusear a impressão dos feedbacks em tela.

Diante disso, é evidente que a plataforma necessita de um desempenho aprimorado, especialmente considerando seu propósito de auxiliar pacientes em recuperação fisioterapêutica e praticantes de atividade física. Para atingir tal desiderato, é necessário que a frequência de coleta de posições corporais seja equiparada, ou ao menos muito próxima, à frequência de processamento dos dados. Este imperativo ressalta a necessidade premente de aprimoramentos, visando ao pleno êxito da plataforma em seu escopo funcional.

6.0.2 Limitações

A primeira limitação está relacionada à disponibilidade de recursos computacionais para a execução do código da plataforma. O escopo do estudo propôs-se a analisar o comportamento da aplicação executada em ambiente de nuvem. No entanto, conforme explicitado nos objetivos, não foi contemplado realizar alterações nas configurações das máquinas que hospedam o website. Implementar tais mudanças demandaria um investimento mais substancial em recursos de computação em nuvem. Dessa forma, ao longo do presente estudo, manteve-se a configuração da máquina fixa com 2 processadores.

6.0.3 Recomendações

Investir tempo na análise aprofundada do comportamento de outras funções existentes na aplicação pode revelar-se altamente valioso, considerando que a plataforma ainda apresenta um desempenho aquém do ideal. Além disso, uma utilização mais investigativa do OpenMP e suas diretivas são capazes de otimizar o resultado obtido.

Num futuro próximo, o aumento do investimento financeiro surge como um fator crucial, visto que pode impactar de diversas maneiras, proporcionando maior flexibilidade na configuração das máquinas que hospedam o website e aprofundando a compreensão da eficiência e escalabilidade da plataforma, aproximando-a do principal objetivo proposto.

A estratégia de utilização de módulos do código em Python em bibliotecas compartilhadas em C pode fazer com que essas funções ainda possam ser mais aproveitadas em relação ao poder de processamento concorrente. Portanto, é recomendado que ao longo dos próximos passos de paralelismo na plataforma, essa avaliação seja levada em consideração para mudar de estratégia.

A análise dos resultados da paralelização com diferentes tamanhos e tipos de amostras pode ser benéfica. Ao examinar o resultado final, observou-se que dois grupos musculares apresentaram um aumento de 1 no número de erros de movimentos não percebidos. Apesar da significativa melhoria geral, é recomendável dedicar tempo para compreender se a implementação da paralelização está gerando essa pequena discrepância nos grupos musculares específicos mencionados.

Diante da necessidade de otimização, é pertinente examinar a plataforma sob uma perspectiva mais abrangente. É fundamental avaliar se o investimento em um maior tempo de paralelização de código é realmente necessário ao utilizar os recursos computacionais mais extensivos, ou se uma simples melhoria na máquina responsável pela execução da plataforma na nuvem seria suficiente. Nesse contexto, é crucial analisar até que ponto a proporção de melhoria na exibição dos feedbacks na tela, que no caso do presente estudo melhorou em 120%, persistiria em seu crescimento ou se atingiria um ponto de saturação de acordo com uma certa quantidade de recursos computacionais.

Como última recomendação, vale destacar a possibilidade de otimização da plataforma com o uso de outras técnicas, como o Edge Computing. Nesse caso, parte do código responsável pelo processamento dos dados poderia estar disponível diretamente no fron-end da aplicação. Como consequência, isso pode vir a diminuir a necessidade de comunicação com o back-end e maximixar o tempo ganho.

REFERÊNCIAS

DAVIS, M. *SnakeViz*. 2023. https://jiffyclub.github.io/snakeviz/. Acesso em 30 de novembro de 2023.

DEVELOPERS, C. for. *Chrome for Developers*. 2017. https://developer.chrome.com/docs/devtools/performance?hl=pt-br. Acesso em 02 de dezembro de 2023.

DEVELOPERS, G. for. *MediaPipe*. 2023. https://developers.google.com/mediapipe. Acesso em 22 de novembro de 2023.

DIGITALOCEAN, L. *Calling C Functions from Python*. 2023. https://www.digitalocean.com/community/tutorials/calling-c-functions-from-python. Acesso em 02 de dezembro de 2023.

ERL RICARDO PUTTINI, Z. M. T. *Cloud Computing: Concepts, Technology Architecture*. 1rd. ed. NJ: Prentice Hall Press, 2013. ISBN 978-0-13-338752-0.

FOUNDATION, M. *Window: requestAnimationFrame() method*. 2023. https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame. Acesso em 02 de dezembro de 2023.

FOUNDATION, P. S. *The Python Profilers*. 2022. https://docs.python.org/3/library/profile.html. Acesso em 01 de dezembro de 2023.

INDUSTRIES, A. *Bipolar Transistor*. 2017. https://www.flickr.com/photos/adafruit/37498202591/in/photostream/. Acesso em 28 de novembro de 2023.

PACHECO, P. *An Introduction to Parallel Programming*. 1rd. ed. San Francisco: Morgan Kaufmann Publishers Inc., 2011. ISBN 978-0-12-374260-5.

PRAKASH, D. *Building a Python C Extension Module*. 2019. https://realpython.com/build-python-c-extension-module/. Acesso em 04 de dezembro de 2023.