A Comparative Analysis of the Smith-Waterman Algorithm on Raspberry Pi-Based Parallel Platforms

Lucas Freire Sêmeler, and Wanderson Roger Azevedo Dias

***Abstract*—Sequence alignment is a fundamental task in bioinformatics, requiring intensive computational processing that grows quadratically with sequence size. High-Performance Computing (HPC) offers essential solutions to accelerate such tasks. This paper presents a detailed performance analysis of the local alignment Smith-Waterman algorithm, comparing a sequential implementation against parallel versions designed for modern multi-core and nodes architectures. For shared-memory parallelism, an OpenMP version was developed using a wave front strategy to manage data dependencies; for distributed-memory, an MPI version was implemented using a 2D row-based domain decomposition. The evaluation results, using workloads of sequences sizes of 1000, 5000, and 15000, revealed distinct performances. The OpenMP approach proved effective for larger workloads (peak speedup of 1.84x), though inefficient for small workloads (speedup of 0.56x) due to parallelization overhead. In contrast, the MPI approach was consistently outperformed by the sequential version in all tests, demonstrating that the high cost of inter-node communication nullified the gains from distributed computing. The analysis concludes that the choice of a parallel model must carefully balance architectural paradigms with algorithmic characteristics to achieve meaningful performance improvements.**

Link to graphical and video abstracts, and to code: https://latamt.ieeer9.org/index.php/transactions/article/view/10088

***Index Terms*—High-Performance Computing, Parallel Computing, Smith-Waterman, Raspberry Pi, Sequence Alignment.**

I. Introdução

A Computação de Alto Desempenho (*High-Performance Computing* - HPC) tem se consolidado como um pilar essencial no avanço científico e tecnológico de diversas áreas, incluindo saúde, bioinformática, engenharias e ciências ambientais [1], [2]. O crescimento exponencial na geração de dados, especialmente em aplicações biológicas e médicas, como o sequenciamento genômico e proteômico de nova geração, impõe desafios computacionais cada vez mais complexos, cuja superação exige arquiteturas de processamento robustas e escaláveis.

The associate editor coordinating the review of this manuscript and approving it for publication was Carlos Thomaz (Corresponding author: *Lucas Sêmeler*).

This work was supported by Federal Institute of Rondônia - IFRO (Public Notice Nº 7/2024/REIT - PROPESP/IFRO - PIBITI - Cycle 2024-2026).

Lucas Sêmeler, and W. R. A. Dias are with Laboratory of Computational Architectures and Parallel Computing, Federal Institute of Rondônia, Ji-Paraná, RO, Brazil (e-mails: lucasemeler@gmail.com, and wradias@gmail.com).

Nesse contexto, o alinhamento de sequências biológicas desponta como uma das tarefas mais críticas da bioinformática, sendo frequentemente utilizada para comparação de genomas, identificação de mutações e inferência de relações evolutivas. Trata-se de um problema inerentemente intensivo em termos computacionais, sobretudo em sua forma exata, representada por algoritmos como o *Smith-Waterman* [3]. Este algoritmo, por ser baseado em programação dinâmica, garante alinhamentos locais ótimos, porém possui complexidade de tempo e espaço - $O(mn)$ - o que o torna inviável para conjuntos de dados de grande escala sem o uso de técnicas de aceleração computacional.

Diante dessa limitação, o uso de paralelismo computacional se torna não apenas uma alternativa, mas uma necessidade. A evolução das arquiteturas de computadores permitiu o surgimento de sistemas multicore, multiprocessadores e *clusters* de computadores interconectados, que habilitam diferentes modelos de paralelismo - como memória compartilhada e memória distribuída - para explorar o desempenho escalável desses algoritmos. A predominância de supercomputadores massivamente paralelos na lista TOP500 [4] ilustra a importância estratégica da HPC na solução de problemas intensivos em computação.

Para explorar esses modelos arquiteturais, *frameworks* como OpenMP (*Open Multi-Processing*) [5] e MPI (*Message Passing Interface*) [6] são amplamente adotados. O OpenMP oferece uma abordagem de paralelismo em memória compartilhada, sendo ideal para ambientes multicore; já o MPI é projetado para ambientes com memória distribuída, como *clusters*, permitindo que múltiplos processos executem em nós distintos e colaborem via troca de mensagens. Esses modelos se complementam e refletem diferentes paradigmas computacionais dentro da HPC.

Nos últimos anos, plataformas de hardware acessíveis como a Raspberry Pi têm sido utilizadas como base para ensino, prototipagem e pesquisa em computação paralela. Apesar de suas limitações de desempenho, o baixo custo, a portabilidade e a flexibilidade dessas plataformas permitiram a formação de *clusters* experimentais, viabilizando estudos práticos em ambientes reais [7], [8], [9], [10].

Neste cenário, o presente artigo apresenta uma análise comparativa de desempenho do algoritmo *Smith-Waterman*, implementado em três abordagens distintas: uma versão sequencial tradicional, uma versão paralela baseada em OpenMP para execução em uma única plataforma de Raspberry Pi (explorando o paralelismo com memória compartilhada), e uma versão baseada em MPI para execução em um *cluster* constituído com cinco nós de Raspberry Pi,

modelo 5 (explorando o paralelismo com memória distribuída). O *cluster* montado foi denominado de *ClusterPi-5* e possui um nó mestre e quatro nós escravos. As avaliações foram realizadas com diferentes tamanhos de sequência como carga de trabalho, permitindo verificar o impacto da arquitetura e do modelo de paralelismo na eficiência do algoritmo.

O restante deste artigo está organizado da seguinte forma: a Seção II apresenta a fundamentação teórica sobre o algoritmo *Smith-Waterman* e os modelos de programação paralela usados; a Seção III apresenta alguns trabalhos correlatos sobre o uso e análises do algoritmo *Smith-Waterman*; a Seção IV detalha a metodologia adotada e o ambiente experimental utilizado; a Seção V apresenta os resultados obtidos e uma análise crítica de desempenho; e, por fim, a Seção VI traz as conclusões e sugestões para trabalhos futuros.

II. Fundamentação Teórica

A. O Algoritmo Smith-Waterman

Proposto por *Temple Smith* e *Michael Waterman* em 1981 [3], o algoritmo *Smith-Waterman* é considerado um dos métodos mais precisos para alinhamento local de sequências biológicas, como cadeias de nucleotídeos (DNA, RNA) ou aminoácidos (proteínas). Baseado em programação dinâmica, o algoritmo visa encontrar a região de maior similaridade entre duas sequências, sem necessariamente alinhar seus extremos, ao contrário do algoritmo de *Needleman-Wunsch*, que realiza alinhamentos globais.

O procedimento consiste na construção de uma matriz de pontuação H de dimensões $(m+1) \times (n+1)$, onde m e n são os comprimentos das sequências $S1$ e $S2$, respectivamente. Cada célula $H(i,j)$ representa a pontuação máxima de um alinhamento local que termina nas posições i e j das respectivas sequências, e é definida pela relação de recorrência:

$$H(i,j) = max \begin{cases} H(i-1,j-1) + w(S1[i-1], S2[j-1]) \\ H(i-1,j) + wgap \\ H(i,j-1) + wgap \\ 0 \end{cases} \quad (1)$$

onde $i,j > 0$.

A matriz é inicializada com zeros nas bordas ($H(i,0) = H(0,j) = 0$ e os valores da função $w(a,b)$ representam a pontuação para uma correspondência (*match*), uma divergência (*mismatch*), ou a penalidade para *gaps*. O termo '0' garante que a pontuação nunca seja negativa, o que caracteriza o alinhamento local: a computação pode ser reiniciada a partir de qualquer célula se os valores anteriores forem desfavoráveis.

Após o preenchimento da matriz, o algoritmo realiza o processo de ***traceback***, que percorre retroativamente as células a partir da posição de maior valor na matriz H, seguindo o caminho que levou à pontuação ótima. Este caminho determina os segmentos de sequências alinhados e os *gaps* inseridos.

A complexidade computacional do algoritmo é $O(mn)$ tanto em tempo quanto em espaço, o que o torna intensivo em recursos computacionais para sequências de grande porte - um fator limitante em muitas aplicações práticas. Além disso, a paralelização eficiente é desafiadora devido às dependências de dados: cada célula depende de três vizinhos imediatamente anteriores (acima, à esquerda e na diagonal superior esquerda). Isso impõe uma ordem estrita de processamento, que inviabiliza paralelismo trivial.

Para contornar esse obstáculo, uma abordagem amplamente adotada é a paralelização por ***wavefront*** (frente de onda), na qual as células de mesma anti-diagonal (isto é, com $i+j = k$ constante) podem ser computadas em paralelo, pois são mutuamente independentes. Essa técnica é particularmente relevante em arquiteturas paralelas, pois permite a exploração eficiente do paralelismo intrínseco do algoritmo. A Fig. 1 apresenta o fluxo de execução do Algoritmo *Smith-Waterman*.

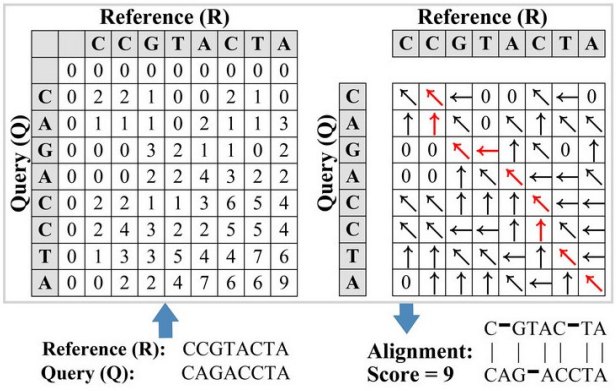


Fig. 1. Fluxo de execução do Algoritmo *Smith-Waterman.*

B. Biblioteca OpenMP

OpenMP (*Open Multi-Processing*) é uma API amplamente utilizada para programação paralela em arquiteturas de memória compartilhada [11]. Lançada em 1997, a especificação define um conjunto de diretivas de compilador, bibliotecas de suporte e variáveis de ambiente que permitem a anotação explícita de regiões paralelas no código-fonte em linguagens como C, C++ e Fortran.

No contexto de arquitetura de computadores, o OpenMP permite a exploração direta de múltiplos núcleos de processamento em um único nó físico, utilizando um modelo de paralelismo baseado em *threads* que compartilham o mesmo espaço de endereçamento de memória. O uso da diretiva #pragma omp parallel delimita regiões do código que serão executadas por múltiplas *threads* simultaneamente [12].

Um dos principais benefícios do OpenMP é a simplicidade de implementação, que torna possível a prototipagem rápida de algoritmos paralelos. No entanto, sua eficácia depende de uma série de fatores arquiteturais, como o número de núcleos disponíveis, o *cache* compartilhado, o acesso à memória RAM

e a afinidade das *threads*. Além disso, o balanceamento de carga entre *threads* pode ser ajustado com a cláusula schedule, influenciando diretamente o desempenho da aplicação.

Embora tradicionalmente utilizado em sistemas com memória compartilhada, o OpenMP pode ser combinado com MPI em ambientes híbridos para aproveitar ao máximo os recursos de *clusters* heterogêneos ou supercomputadores com múltiplos nós e múltiplos núcleos por nó [13].

C. Biblioteca MPI

A interface MPI (*Message Passing Interface*) é o principal padrão de programação paralela para ambientes de memória distribuída [6], [13]. Desenvolvida por um consórcio internacional de pesquisadores e empresas (*MPI Forum*), a biblioteca oferece um conjunto de funções que permitem a comunicação eficiente entre processos independentes que não compartilham espaço de memória, mas trocam dados explicitamente por mensagens.

Diferentemente do OpenMP, o modelo de paralelismo do MPI é baseado na cooperação entre múltiplos processos que podem estar distribuídos em diferentes nós de um *cluster*. Cada processo possui seu próprio espaço de endereçamento e comunica-se com os demais por meio de primitivas como MPI_Send, MPI_Recv, MPI_Bcast, MPI_Scatter, entre outras.

A flexibilidade e a portabilidade do MPI o tornaram o padrão de fato para HPC em plataformas com grande número de nós. Contudo, a complexidade associada à identificação de zonas paralelizáveis, ao gerenciamento de comunicação e à sincronização entre processos representa um desafio significativo para o desenvolvedor. Toda a paralelização deve ser explicitamente codificada, o que exige profundo conhecimento tanto do algoritmo quanto da arquitetura de execução [14].

Existem diversas implementações da especificação MPI, incluindo opções comerciais (Intel MPI, HP MPI) e de código aberto amplamente utilizadas, como OpenMPI e MPICH. A aplicação eficiente do MPI depende de fatores como latência da rede, largura de banda, topologia do *cluster* e estratégia de decomposição de dados adotada - sendo estas variáveis cruciais para o desempenho de algoritmos paralelos.

III. Trabalhos Correlatos

No artigo *Algorithms for Strings and Sequences: Pairwise Alignment* [15] o autor apresentou um estudo sobre o alinhamento pareado entre sequências biológicas, como DNA, RNA e/ou proteínas, sendo considerado um problema clássico e amplamente explorado na bioinformática, estando presente em diversas análises que envolvem dados oriundos de processos de sequenciamento genético. Entre as abordagens mais consolidadas, o autor destaca as versões global e local do problema, respectivamente tratadas pelos algoritmos de *Needleman-Wunsch* e *Smith-Waterman*. Ambos utilizam programação dinâmica para reconstruir alinhamentos ótimos conforme uma função de pontuação predefinida, sendo que o primeiro busca o melhor alinhamento global entre duas sequências completas, enquanto o segundo identifica regiões localmente semelhantes. Além disso, o autor também destaca que diferentes funções de pontuação têm sido propostas e aplicadas na prática para aprimorar a qualidade dos alinhamentos, considerando particularidades estruturais e evolutivas das sequências biológicas. Assim, esses métodos fundamentais constituem a base de muitos sistemas modernos de análise e comparação de sequências, destacando-se como referência essencial nos estudos relacionados ao alinhamento par-a-par.

O artigo *A Review of Parallel Implementations for the Smith-Waterman Algorithm* [16], os autores afirmam que os avanços recentes nas tecnologias de sequenciamento genético resultaram em um volume sem precedentes de dados genômicos, tornando o alinhamento de sequências um problema central em diversas etapas de análise bioinformática, especialmente nas abordagens de alinhamento local, frequentemente baseadas no algoritmo de *Smith-Waterman*. Os autores destacam que embora esse algoritmo ofereça alta precisão na identificação de correspondências ótimas entre subsequências, sua elevada complexidade computacional motivou o desenvolvimento de múltiplas estratégias de aceleração e paralelização, incluindo paralelização em nível de vetor, *threads*, processos e o uso de arquiteturas heterogêneas. Os autores relatam que a literatura atual ainda necessita de sistematização, o que limita o avanço consistente de soluções paralelas e que estudos recentes têm buscado organizar esse panorama, descrevendo diferentes modelos de disposição de dados e destacando a importância do alinhamento em larga escala em comparações genômicas. Assim, avaliações de desempenho usando ferramentas típicas reforçam tendências e desafios atuais, contribuindo para orientar futuras pesquisas e fornecer subsídios técnicos para desenvolvedores e pesquisadores na seleção e no aprimoramento de ferramentas de alinhamento.

Já no artigo *Parallelization of the Smith-Waterman Algorithm to Accelerate DNA Sequence Alignment* [17], os autores destacam que os estudos genômicos impõe novos desafios na comparação de sequências biológicas, sendo uma atividade essencial para identificar similaridades entre sequências genéticas. Nesse contexto, os métodos de alinhamento global e local desempenham papéis complementares, sendo que o alinhamento local, exemplificado pelo algoritmo de *Smith-Waterman*, se mostra particularmente eficiente por restringir a busca a segmentos específicos das sequências. Os autores avaliaram a execução sequencial e paralela do método de *Smith-Waterman*, demonstrando que a paralelização proporciona ganhos expressivos de desempenho, alcançando até 78,5% de melhoria em relação à versão sequencial. Os resultados mostraram que o aumento do tamanho da matriz de alinhamento potencializa ainda mais os benefícios do processamento paralelo, reforçando a importância de abordagens paralelas para lidar com o crescimento contínuo dos dados genômicos em aplicações modernas de bioinformática.

Os autores do artigo *Performance Analysis of Hybrid MPI and OpenMP on Smith-Waterman Algorithm* [18], afirmam

que estudos recentes têm destacado a importância do alinhamento de sequências como tarefa central na bioinformática, especialmente para a análise de informações genéticas e relações evolutivas em cenários de crescimento acelerado dos dados biológicos. Nesse contexto, o algoritmo de *Smith-Waterman* permanece como referência pela precisão que oferece, embora seu custo computacional se torne um desafio em bases de dados de grande porte. Para contornar essa limitação, algumas pesquisas propõem estratégias de paralelização híbrida que combinam MPI e OpenMP, explorando simultaneamente comunicação distribuída e paralelismo em memória compartilhada. Os autores relatam que essas abordagens buscam melhorar a eficiência, escalabilidade e velocidade de execução do algoritmo em sistemas de computação de alto desempenho, possibilitando análises mais rápidas e precisas. Assim, os autores apresentam propostas significativas para o avanço das técnicas de alinhamento local em ambientes HPC que reforçam o papel de modelos híbridos como solução promissora para demandas computacionais cada vez maiores na bioinformática.

No artigo *Performance Improvement of the Parallel Smith Waterman Algorithm Implementation using Hybrid MPI-OpenMP Model* [19], os autores relatam que pesquisas recentes têm explorado modelos paralelos híbridos para otimizar o desempenho do algoritmo *Smith-Waterman* (SW), combinando arquiteturas de memória compartilhada e distribuída por meio do uso conjunto de MPI e OpenMP. O estudo apresenta uma implementação aprimorada do SW que realiza o cálculo da matriz de alinhamento linha a linha, reduzindo significativamente o consumo de memória e garantindo melhor eficiência computacional. Os autores executaram o SW híbrido em um *cluster* homogêneo composto por oito nós e vinte e quatro núcleos, os autores também afirmaram que a abordagem demonstrou excelente escalabilidade e obteve expressivos ganhos de desempenho ao ser aplicada à base de proteínas SWISS-PROT, alcançando *speedups* de até 14x em relação à versão OpenMP e 50x em comparação à versão sequencial. Assim, os resultados evidenciaram o potencial dos modelos híbridos para acelerar algoritmos de alinhamento local em cenários de alto desempenho, reforçando sua relevância no contexto de aplicações em bioinformática e computação paralela.

Os dois últimos estudos analisados apresentam maior proximidade conceitual e metodológica com a pesquisa aqui desenvolvida.

IV. METODOLOGIA

A. Implementações do Algoritmo

Para a realização da análise comparativa proposta, foram desenvolvidas três versões distintas do algoritmo *Smith-Waterman* utilizando a linguagem de programação C, com o objetivo de avaliar o desempenho sob diferentes paradigmas de paralelismo computacional.

- Versão Sequencial: Implementação base do algoritmo, onde a matriz de pontuação *H* é preenchida de forma iterativa, linha por linha, utilizando apenas um núcleo de processamento. Esta versão serve como referência para cálculo de *speedup* e identificação dos impactos do paralelismo.
- Versão OpenMP: A versão sequencial foi paralelizada por meio da API OpenMP, utilizando a diretiva `#pragma omp parallel for`. A estratégia de paralelização adotada foi baseada na abordagem *wavefront*, processando as anti-diagonais da matriz de forma paralela. Essa escolha permite contornar as dependências de dados intrínsecas ao algoritmo. Além disso, foi utilizada a cláusula `schedule` para promover melhor balanceamento de carga entre as *threads*, fundamental para garantir um uso eficiente dos núcleos disponíveis em arquiteturas multicore.
- Versão MPI: Para o ambiente de memória distribuída, foi implementada uma versão utilizando a biblioteca MPI com uma estratégia de decomposição de domínio 2D (em blocos). A matriz de pontuação global é dividida em uma grade de submatrizes, e cada processo MPI é responsável por calcular um bloco. O processo `rank 0` gera as sequências e as distribui para os processos de acordo com sua posição na grade. A comunicação das dependências (bordas dos blocos) é orquestrada em um padrão *wavefront*, onde cada processo só inicia seu cálculo após receber os dados necessários de seus vizinhos de cima, da esquerda e da diagonal superior-esquerda. Para evitar *deadlocks*, a comunicação é realizada de forma não-bloqueante, utilizando as funções `MPI_Irecv`, `MPI_Isend` e `MPI_Waitall`.

Para garantir consistência na avaliação, os parâmetros de pontuação utilizados em todas as versões foram fixados como:

$$\text{MATCH} = +2$$
$$\text{MISMATCH} = -1$$
$$\text{GAP} = -1$$

Essa uniformidade permite isolar o impacto do modelo de paralelismo sobre o desempenho do algoritmo.

B. Ambiente e Procedimento Experimental

Os testes foram realizados com plataforma de baixo custo, ou seja, com Raspberry Pi, modelo 5, processador ARM Cortex-A76 *quad-core* de 2,4 GHz, com 8 GB de memória RAM (ver Fig. 2). O ambiente foi configurado de duas formas:

- Execução *Single-node*: Utilizou-se uma única Raspberry Pi 5 para avaliar a versão sequencial e a versão com OpenMP, explorando o paralelismo de memória compartilhada com 4 *threads* (correspondentes aos quatro núcleos físicos disponíveis na Raspberry Pi) para a versão OpenMP e uma *thread* para a sequencial.
- Execução *Multi-node* (*ClusterPi-5*): Um *cluster* composto por cinco nós com Raspberry Pi 5 interligados por um *switch Gigabit Ethernet* (modelo TP-Link TL-SG116E) e cabos de rede *Ethernet RJ45*

Cat8 (velocidade de 40 Gbps e frequência de 2000 MHz) foram utilizados para a execução da versão MPI, executando um processo por nó. Essa configuração explora o paralelismo com memória distribuída e simula um ambiente típico de HPC com comunicação entre nós físicos.

O sistema operacional usado foi o *Raspberry Pi OS* (*64-bits*), utilizando o compilador GCC (*GNU Compiler Collection*) com suporte às bibliotecas OpenMP e OpenMPI. Essa escolha garante uma base consistente e compatível com o ecossistema de HPC de código aberto.

Para as medições, foram escolhidos três tamanhos de sequências (1000, 5000 e 15000 elementos), representando pequenas, médias e grandes cargas de trabalho, respectivamente. Cada configuração experimental foi executada 100 vezes, e o tempo médio de execução foi registrado com exatidão de milissegundos, buscando reduzir o impacto de variações pontuais e garantir significância estatística nos resultados.

A principal métrica de avaliação de desempenho adotada foi o *Speedup*, definido pela razão:

$$Speedup = \frac{Tempo\ Serial}{Tempo\ Paralelo} \tag{2}$$

onde T_{Serial} é o tempo de execução da versão sequencial e T_{Paralelo} é o tempo da respectiva versão paralela (OpenMP ou MPI). Essa métrica quantifica o ganho de desempenho obtido com a paralelização e permite avaliar a escalabilidade do algoritmo nas diferentes plataformas.

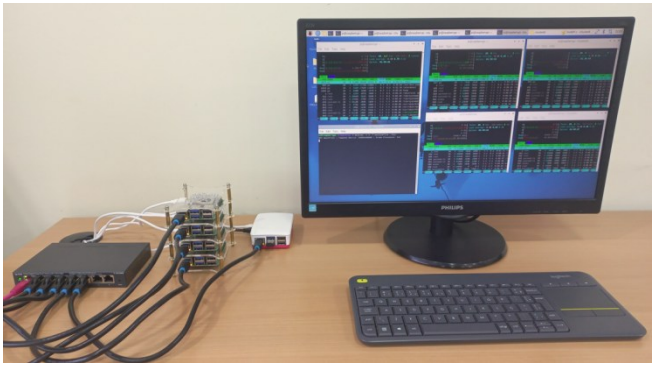


Fig. 2. Setup implantado para a execução dos experimentos.

V. Resultados e Discussão

A análise dos resultados obtidos a partir da execução das três versões do algoritmo *Smith-Waterman* - sequencial, paralela com OpenMP e paralela com MPI - permite uma avaliação aprofundada das implicações do paralelismo em plataformas de baixo custo como a Raspberry Pi. As cargas de trabalho foram deliberadamente selecionadas para representar diferentes níveis de esforço computacional, como 1000, 5000 e 15000 elementos, permitindo identificar limites, vantagens e desvantagens de cada abordagem, e, principalmente, observar o comportamento de escalabilidade em função do tamanho do problema.

Sendo assim, nesta seção são discutidos os resultados, considerando tanto os fatores algorítmicos quanto arquiteturais que influenciam diretamente o desempenho computacional.

A. Relação entre Carga de Trabalho e Eficiência de Paralelização

1) Carga pequena (1000×1000): Na menor carga de trabalho avaliada, a versão sequencial superou ambas as versões paralelas (0,028s). A versão com OpenMP, que utiliza paralelismo em memória compartilhada, obteve *speedup* de apenas 0,56x, ou seja, apresentou desempenho 79% inferior que a versão sequencial, executando em média aproximadamente 0,050s. Essa degradação ocorre por fatores amplamente conhecidos na literatura de HPC:

- O custo de inicialização e gerenciamento das *threads* em OpenMP não é compensado pela quantidade reduzida de operações que o algoritmo realiza em pequenas matrizes. Em termos práticos, a aplicação incorre em *overhead* de criação, sincronização e escalonamento de *threads*, com custo comparável, ou até superior, ao tempo de computação real.
- A estratégia *wavefront*, embora eficaz para evitar violação de dependências de dados, introduz complexidade adicional no controle do fluxo de execução. O cálculo das anti-diagonais requer sincronização entre etapas, o que reduz o potencial de paralelismo em cargas pequenas.

A versão MPI apresentou o pior desempenho absoluto, com tempo de execução quase 100x maior que a versão sequencial, ou seja, um tempo de execução de 2,752s. Isso evidencia um ponto central na análise de algoritmos distribuídos: quando a carga de computação é pequena, o tempo de comunicação entre nós domina completamente o tempo de execução total. Essa comunicação inclui:

- O *broadcast* da sequência completa para todos os processos (MPI_Bcast).
- A distribuição das linhas da matriz (MPI_Scatterv).
- A troca encadeada das fronteiras entre processos adjacentes, devido à dependência entre linhas vizinhas da matriz.

Esse comportamento confirma o que foi apresentado na fundamentação teórica (seção II) sobre a relação entre comunicação e computação nos modelos de memória distribuída. Como cada célula depende das vizinhas imediatamente anteriores, o modelo MPI exige sincronização frequente e bloqueante, o que penaliza fortemente o desempenho, principalmente quando o volume de dados não justifica o paralelismo. A Fig. 3 apresenta os resultados do tempo médio para execução das três versões distintas (sequencial, OpenMP e MPI) da implementação do algoritmo *Smith-Waterman* com carga de trabalho de 1000 elementos.

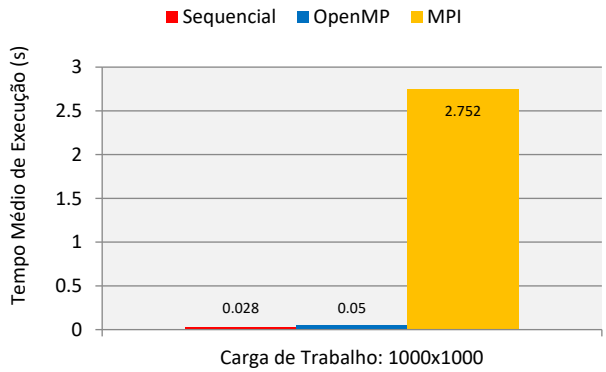


Fig. 3. Desempenho da execução do algoritmo Smith-Waterman com carga de 1000 elementos.

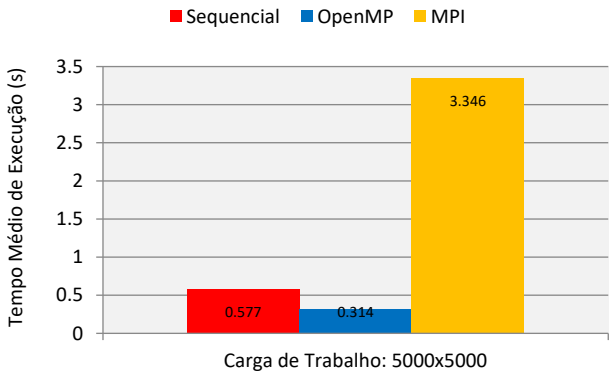


Fig. 4. Desempenho da execução do algoritmo *Smith-Waterman* com carga de 5000 elementos.

2) Carga intermediária (5000×5000): Com o aumento do tamanho do problema, verifica-se o que na literatura de HPC é conhecido como ponto de inflexão da eficiência paralela. A versão OpenMP (0,314s) passa a superar a sequencial (0,577s), com *speedup* de 1,84x, o que indica que o custo fixo de paralelismo passou a ser diluído no tempo total de execução.

Esse cenário é típico de problemas onde a granularidade computacional (ou seja, o tempo de cálculo por unidade de comunicação ou sincronização) começa a justificar o uso de múltiplos núcleos. Nesse ponto, a arquitetura multicore da plataforma Raspberry Pi - ainda que modesta, ou seja, um *clock* de 2,4 GHz - começa a ser melhor aproveitada, pois:

- O número de anti-diagonais é suficientemente grande para permitir que múltiplas *threads* executem por períodos prolongados sem necessidade de sincronização constante.
- O volume de dados carregados por cada *thread* é maior, reduzindo a relação *overhead*/cálculo.

A versão MPI (3,346s), por sua vez, ainda segue como a menos eficiente. O modelo de decomposição 2D (em blocos), utilizado nesta implementação, embora teoricamente mais escalável, também apresentou gargalos de desempenho significativos no ambiente testado. A necessidade de cada processo receber dados de múltiplos vizinhos (superior, esquerdo e diagonal) antes de iniciar seu cálculo, somada à alta latência da comunicação em rede do *cluster*, resulta em um *overhead* considerável. Mesmo com uma estratégia de comunicação não-bloqueante, os processos podem passar uma parte significativa do tempo ocioso, esperando a chegada dos dados necessários para avançar na "onda" de processamento. Isso demonstra que, mesmo com uma estratégia de decomposição mais avançada, o alto custo da comunicação na plataforma testada foi o fator dominante que limitou o desempenho. A Fig. 4 apresenta os resultados do tempo médio para execução das três versões distintas (sequencial, OpenMP e MPI) da implementação do algoritmo *Smith-Waterman* com carga de trabalho de 5000 elementos.

3) Carga grande (15000×15000): Na maior carga de trabalho, observa-se o impacto crescente das limitações arquiteturais da plataforma. Embora a versão OpenMP (3,354s) continue apresentando ganhos de desempenho com relação à sequencial (5,125s), ou seja, *speedup* de 1,53x, houve uma redução em relação à carga intermediária. Esse fenômeno é consistente com o que se observa em sistemas com recursos compartilhados e limitada largura de banda de memória:

- As quatro *threads* concorrem intensamente pelo acesso à RAM, saturando o barramento de memória.
- O aumento da matriz reduz a eficiência do uso de *cache*, elevando o número de *cache misses* e penalizando o tempo de acesso à memória principal.
- O sistema operacional pode incorrer em trocas de contexto e preempções, uma vez que os núcleos são compartilhados e há concorrência por recursos.

A versão MPI (7,851s), por sua vez, apresentou melhoria relativa (*speedup* de 0,65x), mas ainda insuficiente para superar sequer a versão sequencial. Esse pequeno ganho pode ser atribuído ao fato de que, com uma matriz maior, o tempo gasto em computação começa a crescer mais rapidamente do que o tempo de comunicação. No entanto, os gargalos da comunicação bloqueante, a latência da rede e a limitação da decomposição 2D ainda impedem qualquer ganho de desempenho real. Isso apresenta o princípio de que a eficiência paralela não cresce indefinidamente com o tamanho do problema; ela depende de um balanceamento ideal entre arquitetura, volume de dados e modelo de paralelismo.

A Fig. 5 apresenta os resultados do tempo médio para execução das três versões distintas (sequencial, OpenMP e MPI) da implementação do algoritmo *Smith-Waterman* com carga de trabalho de 15000 elementos. A Tabela 1 apresenta o *Speedup* das versões paralelas para as três cargas de trabalho.

B. Avaliação dos Modelos de Paralelismo

Com base nos resultados experimentais e nas análises teóricas, é possível traçar uma comparação crítica entre os dois modelos de paralelismo testados.

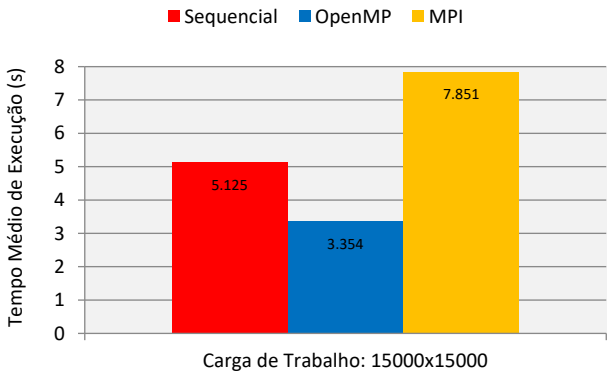


Fig. 5. Desempenho da execução do algoritmo *Smith-Waterman* com carga de 15000 elementos.

TABELA I

SPEEDUP DAS VERSÕES PARALELAS

Carga de Trabalho	*Speedup* OpenMP	*Speedup* MPI
1000	0,56x	0,01x
5000	1,84x	0,17x
15000	1,53x	0,65x

1) OpenMP (memória compartilhada):

- Beneficia-se da baixa latência de comunicação entre *threads* e do acesso direto à memória comum.
- Apresenta melhor desempenho em cargas moderadas e grandes, desde que a arquitetura ofereça *cache* suficiente e largura de banda compatível.
- É limitado por contenção de memória e escala mal em arquiteturas com poucos núcleos físicos ou em cargas extremamente grandes, devido à competição por recursos.

2) MPI (memória distribuída):

- É escalável em arquiteturas com rede de alta largura de banda e baixa latência (como rede *Infiniband*).
- No ambiente testado (*ClusterPi-5*), sofreu penalizações severas devido à alta latência de rede e à escolha de decomposição 2D.
- Sua efetividade está diretamente ligada à eficiência da estratégia de decomposição e à granularidade do problema.

Essa análise corrobora a afirmação de que a escolha do modelo de paralelismo deve considerar não apenas a estrutura algorítmica, mas também as características específicas da arquitetura de hardware e da infraestrutura da rede de interconexão.

C. Contribuições da Plataforma Raspberry Pi

A escolha da plataforma Raspberry Pi, modelo 5, como base tecnológica experimental neste artigo, permitiu avaliar não apenas a viabilidade de executar algoritmos de alto custo computacional em plataformas de baixo custo financeiro, mas também os limites físicos e computacionais desse tipo de solução.

- O uso de OpenMP mostrou que, mesmo com recursos modestos, é possível obter ganhos reais de desempenho em problemas de interesse científico, desde que se respeitem os limites arquiteturais da plataforma.
- Por outro lado, o uso de MPI em um *cluster* de Raspberry Pi evidenciou o impacto severo do *overhead* de comunicação e a importância de redes de baixa latência para que o paralelismo distribuído seja eficaz.

Essas observações são valiosas para o ensino, a prototipagem de soluções paralelas e para a pesquisa em ambientes restritos, nos quais o acesso a supercomputadores é limitado. A plataforma Raspberry Pi, portanto, se mostra adequada como ferramenta de validação funcional e didática, embora tenha restrições importantes quanto à sua aplicabilidade em cenários de produção de larga escala.

Ainda vale destacar que no *ClusterPi-5*, a comunicação entre os nós, mesmo usando um *switch Gigabit* e os cabos de rede *Ethernet RJ45 Cat8*, não superou o *overhead* inerente da implementação MPI, que é um fator crítico para algoritmos com comunicação frequente ou serializada. Conforme observado em outros estudos com *clusters* de Raspberry Pi [7], [20], o *overhead* de comunicação em MPI pode facilmente anular os ganhos do processamento distribuído, especialmente para algoritmos que não são "embaraçosamente paralelos".

Em suma, a análise dos resultados reforça a conclusão de que, para o algoritmo *Smith-Waterman* neste ambiente, a paralelização com OpenMP é uma estratégia viável para cargas de trabalho suficientemente grandes. A abordagem MPI, por outro lado, exigiria estratégias de decomposição mais avançadas (como 2D) e, possivelmente, uma rede de interconexão de menor latência para se tornar competitiva.

VI. CONCLUSÕES

Este trabalho apresentou uma análise de desempenho comparativa entre três versões do algoritmo *Smith-Waterman* - sequencial, paralela com OpenMP e paralela com MPI - executadas em uma plataforma de baixo custo composta por Raspberry Pi, modelo 5. O objetivo foi investigar o impacto de diferentes modelos de paralelismo sobre o desempenho de um algoritmo bioinformático intensivo, considerando tanto o aspecto algorítmico quanto arquitetural, dentro do contexto de Computação de Alto Desempenho (HPC).

Os resultados obtidos evidenciam que a paralelização com OpenMP representa uma estratégia eficiente e viável para acelerar o alinhamento local de sequências, desde que o volume de dados seja suficiente para justificar o custo de paralelismo. Verificou-se um ponto de inflexão - observado na carga intermediária (5000x5000) - a partir do qual os ganhos obtidos superam o *overhead* de criação e gerenciamento de *threads*, atingindo *speedup* de até 1,84x. No entanto, esse ganho não é linear com o crescimento da carga, o que evidencia os limites impostos pela arquitetura de memória

compartilhada, como a competição por largura de banda e a degradação da localidade de *cache*.

Por outro lado, a versão com MPI não demonstrou ganhos de desempenho em nenhum dos cenários testados. A abordagem baseada em decomposição de domínio 2D e comunicação bloqueante encadeada mostrou-se inadequada para o algoritmo *Smith-Waterman*, cujo padrão de dependência entre células impõe uma ordem estrita de processamento. O padrão de dependência do algoritmo *Smith-Waterman* exige que cada processo aguarde dados de múltiplos vizinhos (superior, esquerdo e diagonal) antes de poder computar seu bloco, criando pontos de espera significativos. O alto custo de comunicação interprocessos, combinado à latência da rede *Gigabit* do *ClusterPi-5*, resultou em tempos de execução superiores até mesmo à versão sequencial, com *speedups* abaixo de 1x em todos os testes.

Esses resultados reforçam um princípio-chave da HPC, ou seja, não existe uma solução única e universal para a paralelização de algoritmos, especialmente em plataformas restritas. A escolha entre modelos de memória compartilhada e distribuída deve ser cuidadosamente orientada por: estrutura de dependência dos dados no algoritmo; granularidade computacional da carga de trabalho; e características específicas do hardware (capacidade de paralelismo, memória, rede).

No caso do *Smith-Waterman*, - um algoritmo com dependência de dados forte e padrões de acesso regulares, porém altamente acoplados -, a paralelização com OpenMP se mostrou mais adequada para a arquitetura da plataforma Raspberry Pi, modelo 5. A implementação com MPI, embora funcional, requer ajustes significativos para ser eficiente, como o uso de estratégias de decomposição 2D e métodos não bloqueantes de comunicação.

Adicionalmente, a avaliação realizada demonstra que a plataforma Raspberry Pi, embora limitada em termos de poder de processamento, representa uma alternativa didática e economicamente acessível para pesquisa e ensino em HPC e computação paralela. A capacidade de prototipar, medir e visualizar o impacto de diferentes estratégias de paralelismo torna o ambiente especialmente útil em contextos acadêmicos ou de iniciação científica.

Como ideias para trabalhos futuros, destacamos: (i) adoção de decomposição 2D na versão MPI, de modo a reduzir as barreiras de sincronização e melhorar a relação computação/comunicação; (ii) desenvolvimento de uma abordagem híbrida MPI + OpenMP, explorando paralelismo em múltiplos níveis (entre nós e entre núcleos), o que pode melhorar o aproveitamento de *clusters* com múltiplos núcleos por nó; (iii) uso de instruções SIMD (*Single Instruction, Multiple Data*) nas versões sequencial e OpenMP, com o objetivo de explorar o paralelismo em nível de dados disponível na arquitetura ARM da plataforma Raspberry Pi; (iv) portabilidade do algoritmo para arquiteturas heterogêneas, como GPUs, por meio de *frameworks* como CUDA ou OpenCL, possibilitando comparação com abordagens baseadas em CPU; (v) avaliação com dados reais e complexos, incluindo sequências genéticas em formato FASTA e matrizes de pontuação com penalidades de *gap* afim, para verificar a escalabilidade do algoritmo em contextos reais de bioinformática; e (vi) análise de eficiência energética, uma métrica fundamental em HPC, especialmente quando se utilizam plataformas de baixo consumo como a Raspberry Pi, para avaliar a relação entre desempenho e gasto energético das abordagens paralelas.

Portanto, através desta investigação, contribui-se para o entendimento prático e teórico dos impactos da paralelização em algoritmos clássicos da bioinformática e evidencia-se o valor de plataformas acessíveis como vetores de experimentação e difusão do conhecimento em HPC.

AGRADECIMENTOS

Os autores agradecem ao Instituto Federal de Rondônia (IFRO), pelo apoio financeiro concedido através do Edital Nº 7/2024/REIT - PROPESP/IFRO - PIBITI - Ciclo 2024-2026, que proporcionou a execução desta pesquisa.

REFERÊNCIAS

[1] W. P. Petersen and P. Arbenz, Introduction to Parallel Computing: A Practical Guide with Examples in C. New York, EUA: Oxford University Press, 1st edition, 2004, isbn: 978-0-19-851576-0, doi: https://doi.org/10.1093/oso/9780198515760.002.0001.

[2] C. Xavier, R. Sachetto, V. Vieira, R. W. Santos, and W. Meira Jr., "Multi level Parallelism in the Computational Modeling of the Heart," in *Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Gramado, RS, Brazil, 2007, pp. 3-10, doi: https://doi.org/10.1109/SBAC-PAD.2007.19.

[3] T. F. Smith and M. S. Waterman, "Identification of Common Molecular Subsequences," Journal of Molecular Biology, vol. 147, no. 1, pp. 195-197, 1981, doi: https://doi.org/10.1016/0022-2836(81)90087-5.

[4] TOP500.org, "Top 500 Supercomputing Site," 2025. [Online]. Available: https://top500.org/.

[5] OpenMP Architecture Review Board, "The OpenMP API Specification for Parallel Programming," 2025. [Online]. Available: http://openmp.org/.

[6] MPI Forum, "MPI: A Message-Passing Interface Standard, Version 4.1," 2025. [Online]. Available: https://www.mpi-forum.org/docs/.

[7] F. A. Lima, W. R. A. Dias, and E. D. Moreno, "Implementação de um Cluster Embarcado usando a Plataforma Raspberry Pi," in *Escola Regional de Alto Desempenho do Rio de Janeiro (ERAD-RJ)*, Nova Iguaçu, RJ, Brasil, 2020, pp. 11-15, doi: https://doi.org/10.5753/eradrj.2020.14509.

[8] A. L. J. Ignácio and W. R. A. Dias, "Análise do Desempenho Computacional de Algoritmos Paralelizados com OpenMP e MPI Executados em Raspberry Pi," in *Workshop de Iniciação Científica - Simpósio em Sistemas Computacionais de Alto Desempenho (SSCAD)*, Porto Alegre, RS, Brasil, 2023, pp. 41-48, doi: https://doi.org/10.5753/wscad_estendido.2023.235967.

[9] G. O. Ribeiro, L. F. Sêmeler, and W. R. A. Dias, "Avaliação de Desempenho dos Algoritmos de Números Primos e Monte Carlo em Ambientes HPC," in *Workshop de Iniciação Científica - Simpósio em Sistemas Computacionais de Alto Desempenho (SSCAD)*, São Carlos, SP, Brasil, 2024, pp. 65-72, doi: https://doi.org/10.5753/sscad_estendido.2024.244768.

[10] L. F. T. Silva and W. R. A. Dias, "Avaliação de Algoritmos de Ordenação de Dados em Ambiente de HPC com Raspberry Pi,"

in *Workshop de Iniciação Científica - Simpósio em Sistemas Computacionais de Alto Desempenho (SSCAD)*, Bonito, MS, Brasil, 2025, pp. 57-64, doi: https://doi.org/10.5753/sscad_estendido.2025.15942.

[11] B. Chapman, G. Jost, and R. van der Pas, Using OpenMP: Portable Shared Memory Parallel Programming. Cambridge, MA, USA: MIT Press, 1st edition, 2007, isbn: 978-0-26-253302-7, doi: https://doi.org/10.5860/choice.46-0930.

[12] C. Xavier, R. Sachetto, V. Vieira, R. W. Santos, and W. Meira Jr., "Multi level Parallelism in the Computational Modeling of the Heart," in *Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Gramado, RS, Brazil, 2007, pp. 3-10, doi: https://doi.org/10.1109/SBAC-PAD.2007.19.

[13] P. S. Pacheco and M. Malensek, An Introduction to Parallel Programming. Morgan Kaufmann, 2nd edition, 2021, isbn: 978-0-12-804605-0, doi: https://doi.org/10.1016/C2015-0-01650-1.

[14] R. Trobec, B. Slivnik, P. Bulić, and B. Robič, Introduction to Parallel Computing: From Algorithms to Programming on State-of-the-Art Platforms. Springer, 1st edition, 2018, isbn: 978-3-319-98833-7, doi: https://doi.org/10.1007/978-3-319-98833-7.

[15] S. Beretta, "Algorithms for Strings and Sequences: Pairwise Alignment," Encyclopedia of Bioinformatics and Computational Biology, vol. 1, no. 1, pp. 22-29, 2019, doi: https://doi.org/10.1016/B978-0-12-809633-8.20317-8.

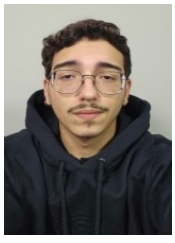
[16] Z. Xia, Y. Cui, A. Zhang, T. Tang, L. Peng, C. Huang, C. Yang, and X. Liao, "A Review of Parallel Implementations for the Smith–Waterman Algorithm," Interdisciplinary Sciences: Computational Life Sciences, vol. 14, no. 1, pp. 1-14, 2021, doi: https://doi.org/10.1007/s12539-021-00473-0.

[17] A. Dhar, I. Chauhan, H. K. Azad, A. Bamzai, and T. Yadav, "Parallelization of the Smith-Waterman Algorithm to Accelerate DNA Sequence Alignment," in *3rd International Conference on Artificial Intelligence For Internet of Things (AIIoT)*, Vellore, India, 2024, pp. 887-892, doi: https://doi.org/10.1109/AIIoT58432.2024.10574609.

[18] K. Ninama, J. Patel, K. K. Girish; M. R. V. S. R. S. Reddy, and B. Bhowmik, "Performance Analysis of Hybrid MPI and OpenMP on Smith-Waterman Algorithm," in *3rd International Conference on Intelligent Systems, Advanced Computing and Communication (ISACC)*, Silchar, India, 2025, pp. 887-892, doi: https://doi.org/10.1109/ISACC65211.2025.10969164.

[19] H. Khaled, H. M. Faheem, M. Fayez, I. Katib, and N. R. Aljohani, "Performance Improvement of the Parallel Smith Waterman Algorithm Implementation using Hybrid MPI-OpenMP Model," in *SAI Computing Conference (SAI)*, London, United Kingdom, 2016, pp. 1232-1234, doi: https://doi.org/10.1109/SAI.2016.7556136.

[20] F. A. Lima, E. D. Moreno, and W. R. A. Dias, "Performance Analysis of a Low Cost Cluster with Parallel Applications and ARM Processors," IEEE Latin America Transactions, vol. 14, no. 11, pp. 4591-4596, 2016, doi: https://doi.org/10.1109/TLA.2016.7795834.



Lucas Freire Sêmeler is an undergraduate student in the Systems Analysis and Development Technology Program at the Federal Institute of Rondônia (IFRO) - Ji-Paraná Campus. He is a member of the Laboratory of Computational Architectures and Parallel Computing (LACCP) and the Research Group on Computational Architectures, Parallelism, and Technological Innovation (GP-ACPIT), where he researches various computational solutions to problems in bioinformatics, using Parallel and High-Performance Computing.

Wanderson Roger Azevedo Dias received his B.Sc. degree in Information Systems (2004) from the Lutheran University Center of Ji-Paraná (CEULJI/ULBRA), and his M.Sc. (2009) and Ph.D. (2013) degrees in Computer Science from the Institute of Computing (ICOMP) at the Federal University of Amazonas (UFAM). He is a professor and researcher at the Federal Institute of Rondônia (IFRO), Ji-Paraná Campus, where he serves as Coordinator of the Laboratory of Computational Architectures and Parallel Computing (LACCP) and as Leader of the Research Group on Computational Architectures, Parallelism, and Technological Innovation (GP-ACPIT). His research interests include Computer Architecture and Organization, Embedded Systems, High-Performance Computing, Parallel Programming, and the Internet of Things. He currently serves as the North 3 Regional Secretary of the Brazilian Computer Society (SBC).