Análise de Desempenho do Algoritmo BLAST de Alinhamento de Sequências Genéticas com OpenMP Executado em Raspberry Pi

Lucas Freire Sêmeler, Wanderson Roger Azevedo Dias

Coordenadoria do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas (CCSTADS)
Laboratório de Arquiteturas Computacionais e Computação Paralela (LACCP)
Instituto Federal de Rondônia (IFRO)
Ji-Paraná – RO – Brasil

{lucasemeler, wradias}@gmail.com

Resumo. *O alinhamento de sequências biológicas é uma tarefa fundamental em bioinformática, mas seu custo computacional cresce rapidamente com o aumento dos dados genômicos. Este trabalho apresenta uma análise de desempenho de uma implementação simplificada do algoritmo BLAST, comparando versões sequencial e paralela (com a biblioteca OpenMP), executadas em um Raspberry Pi 5. Foram utilizadas sequências de DNA de 50.000 a 300.000 pares de bases, e os resultados mostraram que a versão paralela obteve ganhos significativos, alcançando um speedup máximo de 3,81x. Esses resultados demonstram que o paralelismo em memória compartilhada é uma estratégia eficiente para acelerar o BLAST, demonstrando o potencial da computação de alto desempenho em plataformas acessíveis de baixo custo.*

1. Introdução

A Computação de Alto Desempenho (HPC) é fundamental para o avanço científico em áreas que exigem grande capacidade de processamento, como bioinformática, modelagem climática e engenharias, por permitir a solução eficiente de problemas complexos e o tratamento de grandes volumes de dados (Hennessy e Patterson, 2019). Na bioinformática, o aumento exponencial dos dados genômicos tornou o alinhamento de sequências uma tarefa essencial e altamente custosa em termos computacionais (Mount, 2004). Embora algoritmos exatos, como o *Smith-Waterman*, garantam resultados ótimos, sua complexidade quadrática inviabiliza o uso em bases extensas (Smith e Waterman, 1981). Para mitigar esse problema, heurísticas como o BLAST (*Basic Local Alignment Search Tool*) foram propostas, equilibrando sensibilidade e velocidade ao empregar a estratégia de "semeadura e extensão" (*seed-and-extend*) (Altschul *et al.*, 1990).

A busca por maior eficiência no processamento desses algoritmos tem impulsionado o uso de técnicas de paralelismo em arquiteturas modernas, como sistemas *multi-core* e distribuídos (Ignácio e Dias, 2023). Nesse contexto, plataformas acessíveis como a Raspberry Pi (RPi) vêm sendo exploradas em atividades de ensino, prototipagem e pesquisa em HPC. Apesar de suas restrições de desempenho, seu baixo custo e flexibilidade tornam possível a execução de experimentos práticos em computação paralela, oferecendo um ambiente adequado para estudos de desempenho e escalabilidade em sistemas com recursos limitados (Upton e Halfacree, 2013).

Dando continuidade às pesquisas desenvolvidas nessa área, o presente trabalho concentra-se na análise de desempenho de uma implementação paralela do algoritmo BLAST executando em uma RPi. O objetivo é quantificar o ganho de desempenho obtidos com uma versão paralela desenvolvida em OpenMP, em comparação com sua versão sequencial, contribuindo assim para o entendimento da viabilidade e eficiência de soluções HPC acessíveis. O restante do artigo está organizado da seguinte forma: a Seção 2 apresenta a metodologia utilizada na execução da pesquisa; a Seção 3 apresenta os resultados e discussões; e a Seção 4 finaliza com as conclusões, e ideias para trabalhos futuros.

2. Metodologia

Na análise comparativa, empregaram-se duas implementações de uma versão simplificada do algoritmo BLAST, ambas desenvolvidas em linguagem C.

2.1. Descrição do Algoritmo

O algoritmo implementado segue a heurística *seed-and-extend* para alinhamentos locais não espaçados (*ungapped*), estratégia amplamente usada em ferramentas de busca de similaridade biológica, como o BLAST (Altschul *et al.*, 1990; Mount, 2004).

A execução é dividida em três fases principais: **Indexação**, **Busca** e **Extensão**. Na fase de extensão, é aplicado um sistema de pontuação simples (MATCH = +2, MISMATCH = −1) juntamente com a heurística de *X-dropoff* (XDROP = 12), que descarta alinhamentos improdutivos e contribui para a otimização do processo ao reduzir o número de extensões desnecessárias (Altschul *et al.*, 1990).

2.2. Implementação Sequencial

A versão sequencial implementa o algoritmo em um único fluxo de execução, servindo como linha de base (*baseline*) para as análises comparativas de desempenho. Essa abordagem permite avaliar o impacto da paralelização e quantificar o ganho de eficiência obtido pela versão paralela, prática comum em estudos de desempenho de algoritmos bioinformáticos (Parhami, 2006).

2.3. Implementação Paralela com OpenMP

A versão paralela foi desenvolvida utilizando a biblioteca OpenMP, com o objetivo de explorar o paralelismo em arquiteturas de memória compartilhada (Chapman, Jost e Pas, 2007).

A diretiva `#pragma omp parallel for schedule(static)` foi aplicada ao laço principal de busca, que constitui a parte mais intensiva do algoritmo em termos computacionais. Para assegurar a correção dos resultados, cada *thread* mantém um resultado local durante a execução, e, ao final, uma região crítica (`#pragma omp critical`) é utilizada para consolidar de forma segura o melhor alinhamento global. Essa estratégia minimiza *overheads* de sincronização e preserva a consistência dos dados durante a execução paralela.

2.4. Ambiente Experimental

Os testes foram realizados em uma Raspberry Pi, modelo 5 (RPi 5), com processador ARM Cortex-A76 *quad-core* de 2.4GHz e 8GB de memória RAM. O sistema operacional utilizado foi o Raspberry Pi OS (64 *bits*), e a compilação foi realizada com o GCC (*GNU Compiler Collection*), com suporte à biblioteca OpenMP. A versão paralela foi configurada para utilizar 4 *threads*, correspondendo aos quatro núcleos físicos disponíveis no processador da RPi. O ganho de desempenho foi medido através do *speedup*, calculado pela razão:

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

Onde o Tempo Serial representa o tempo de execução sequencial, enquanto o Tempo Paralelo refere-se ao tempo da versão paralela executando em OpenMP. O *Speedup* foi usado para quantificar a melhoria de performance e analisar escalabilidade.

As cargas de trabalho consistiram em sequências de DNA geradas aleatoriamente com quatro tamanhos distintos: 50.000, 100.000, 200.000 e 300.000 pares de bases. Para cada cenário, foram realizadas 100 execuções independentes a fim de garantir relevância estatística, e o tempo médio de execução foi considerado como valor de referência.

3. Resultados e Discussões

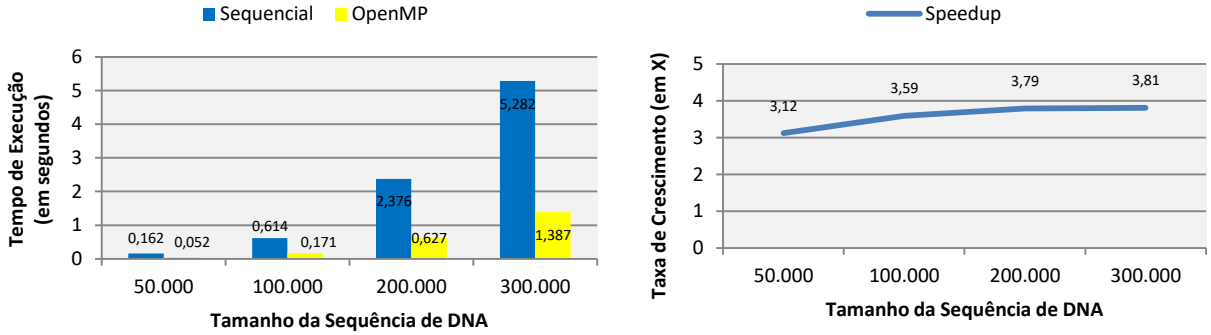
A análise dos resultados apresenta o comportamento do desempenho de duas implementações – sequencial e paralela – considerando as características arquiteturais e limitações da plataforma Raspberry Pi 5. A Tabela 1 apresenta os tempos médios de execução (em segundos) e o desvio padrão, obtidos para diferentes tamanhos de sequência (N), o *speedup* e a eficiência, enquanto a Figura 1(a) ilustra a variação desses tempos entre as duas abordagens e a Figura 1(b) o *speedup* para a versão em OpenMP.

Como esperado, o tempo de execução cresce proporcionalmente ao aumento do tamanho da sequência, refletindo a complexidade linear do algoritmo simplificado. Contudo, observa-se uma redução expressiva do tempo total na versão paralela, evidenciando que a exploração de múltiplos núcleos via OpenMP proporcionou ganhos significativos de desempenho em todos os cenários

avaliados. A diferença entre as curvas de desempenho é consistente e se acentua à medida que a carga computacional aumenta, o que indica boa escalabilidade da abordagem paralela.

Tabela 1. Tempo médio de execução, desvio padrão, *speedup* e eficiência do algoritmo BLAST

Tamanhos de Sequência (*N*)	**Sequencial**	**OpenMP** *4 threads*	***Speedup***	**Eficiência**
	Segundos – Desvio Padrão (s)			
50.000	0,162 – 0,003	0,052 – 0,003	3,12x	78%
100.000	0,614 – 0,007	0,171 – 0,005	3,59x	89,8%
200.000	2,376 – 0,010	0,627 – 0,013	3,79x	94,8%
300.000	5,282 – 0,012	1,387 – 0,042	3,81x	95,2%



(a) tempo de execução (b) *speedup*

Figura 1. Tempo médio de execução e *speedup* do algoritmo BLAST

3.1. Eficiência do Paralelismo

A versão paralela apresentou *speedup* superior a 3× em todas as cargas de trabalho, atingindo 3,81× no maior conjunto de dados testado. Esses resultados confirmam a eficiência do modelo de paralelismo de dados adotado, uma vez que o algoritmo BLAST permite a divisão independente das tarefas de busca entre as *threads*. O uso da diretiva #pragma omp parallel for schedule(static) mostrou-se adequado, pois o balanceamento de carga entre os núcleos foi praticamente uniforme, minimizando o tempo ocioso e os custos de sincronização.

De acordo com Chapman, Jost e Pas (2007), o OpenMP apresenta ótimo desempenho em aplicações de memória compartilhada quando o paralelismo é granular e as operações são independentes, como no caso do processo de busca em bancos de sequências. Assim, os resultados obtidos confirmam a eficiência computacional da paralelização mesmo em um ambiente de hardware modesto.

3.2. Escalabilidade e Limitações Arquiteturais

A análise do *speedup* revela um crescimento progressivo de 3,12× para 3,81× com o aumento do tamanho do problema (ver Figura 1b), indicando que o custo de criação e sincronização de *threads* é amortizado à medida que a carga de trabalho cresce, o que melhora a eficiência do paralelismo. Esse comportamento é típico em sistemas *multi-core*, nos quais o *overhead* inicial tem maior impacto em tarefas menores. Contudo, observa-se uma tendência à saturação próxima de 3,8×, valor ligeiramente inferior ao limite teórico de 4× esperado para um processador *quad-core*, evidenciando limitações práticas da arquitetura ARM Cortex-A76 e do modelo de memória compartilhada.

Entre os principais fatores que restringem a escalabilidade estão a contenção no barramento de memória, as limitações de *cache* L2 e L3 e o *overhead* do sistema operacional, além de restrições térmicas que podem reduzir o *clock* sob cargas elevadas (Hennessy e Patterson, 2019). Ainda assim, os resultados comprovam que a RPi 5 é uma plataforma viável para experimentos em Computação de Alto Desempenho (HPC) de baixo custo, demonstrando que o uso de OpenMP oferece ganhos

expressivos de velocidade e eficiência em aplicações intensivas, validando seu potencial para ensino e pesquisa em ambientes com recursos limitados.

3.3. Eficiência Paralela da versão OpenMP

A análise da eficiência paralela da versão OpenMP demonstra que o desempenho do algoritmo melhora significativamente com o aumento da carga de trabalho, alcançando valores próximos de 95% de eficiência para as maiores sequências testadas (ver Tabela 1). Esse comportamento evidencia que o *overhead* de paralelização – decorrente da criação e sincronização das *threads* – é relevante apenas em problemas menores, sendo rapidamente amortizado à medida que o tamanho das sequências cresce. A eficiência próxima ao limite teórico indica que a divisão das tarefas foi equilibrada entre os quatro núcleos do processador, garantindo alta taxa de ocupação dos recursos computacionais e reduzindo o tempo ocioso entre as *threads*.

Esses resultados confirmam que o modelo de paralelismo de dados do BLAST se adapta de forma eficiente ao ambiente de memória compartilhada oferecido pelo OpenMP e à arquitetura ARM Cortex-A76 da RPi 5. Mesmo considerando limitações inerentes, como a contenção no barramento de memória e o tamanho reduzido de *cache*, a implementação obteve ganhos quase lineares em relação ao número de núcleos utilizados. Assim, a elevada eficiência observada comprova que a paralelização foi corretamente projetada e que a plataforma RPi pode ser empregada com sucesso em experimentos de Computação de Alto Desempenho (HPC) de baixo custo e alto valor didático, especialmente em aplicações bioinformáticas que apresentam independência entre as operações.

4. Conclusões e Trabalhos Futuros

Este trabalho mostrou que a paralelização do algoritmo BLAST com OpenMP executando em uma Raspberry Pi é viável e eficiente, alcançando *speedups* próximos ao limite teórico graças ao alto paralelismo de dados, o que evidencia que dispositivos compactos podem ser usados com sucesso em estudos de HPC quando o paralelismo é bem explorado. A análise destacou que a eficiência depende da compatibilidade entre o modelo de programação e a arquitetura, sendo o OpenMP particularmente adequado à arquitetura ARM Cortex-A76, garantindo execução estável e escalável. Assim, a RPi 5 se consolida como ferramenta acessível para ensino e pesquisa em HPC, e propõem-se como trabalhos futuros a extensão da abordagem para GPUs, a análise de eficiência energética e o estudo de versões mais complexas do BLAST, como os alinhamentos com lacunas (*gapped alignments*).

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

Altschul, S. F.; Gish, W.; Miller, W.; Myers, E. W.; Lipman, D. J. (1990) “Basic Local Alignment Search Tool”. *Journal of Molecular Biology*, 215(3):403-410.

Chapman, B.; Jost, G.; Pas, R. V. D. (2007) “Using OpenMP: Portable Shared Memory Parallel Programming”. Cambridge, Massachusetts, EUA: MIT Press, 1st edition, 353p.

Ignácio, A. L. J.; Dias, W. R. A. (2023) “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, pp. 41-48.

Hennessy, J. L.; Patterson, D. A. (2019) “Arquitetura de Computadores: uma Abordagem Quantitativa”. São Paulo, SP: GEN LTC, 6ª edição, 816p.

Mount, D. W. (2004) “Bioinformatics: Sequence and Genome Analysis”. Laurel Hollow, New York, EUA: Cold Spring Harbor Laboratory Press, 2nd edition, 665p.

Parhami, B. (2006) “Introduction to Parallel Processing - Algorithms and Architectures”. New York, USA: Kluwer Academic Publishers, 1st edition, 532p.

Smith, T. F.; Waterman, M. S. (1981) “Identification of Common Molecular Subsequences”. *Journal of Molecular Biology*, 147(1):195-197.

Upton, E., Halfacree, G. (2013) “Raspberry Pi Manual do Usuário”. São Paulo, Brasil: Novatec, 269p.