Comparação paralela de sequências biológicas longas agregando múltiplos laboratórios de pesquisa

Fernando Machado Mendonça
Departamento de Ciência da Computação
Universidade de Brasília
fernando_cic@aluno.unb.br

Alba Cristina Magalhães Alves de Melo
Departamento de Ciência da Computação
Universidade de Brasília
albamm@cic.unb.br

***Resumo*—A comparação de duas sequências biológicas é uma das operações mais fundamentais da Biologia Computacional pois visa determinar o quão similar duas sequências são. O uso de métodos exatos em tal comparação exige alto poder computacional e uso de grande quantidade de memória. Por essa razão, normalmente clusters de computadores são usados nessa comparação. Neste artigo, propomos e avaliamos uma variante paralela para comparação exata de sequências biológicas longas que se executa em clusters de computadores que envolvam múltiplos laboratórios de pesquisa. Uma estratégia é proposta para que se consiga o uso mais balanceado das máquinas que compõem o cluster, reduzindo o tempo ocioso e a comunicação entre as mesmas. Com o uso de máquinas já existentes em laboratórios de pesquisa de nossa Universidade durante os finais de semana, construímos uma plataforma de computação de alto desempenho de custo zero, envolvendo 2 laboratórios e um total de 128 processadores. Os resultados obtidos nessa plataforma mostram que, na comparação de duas sequências de aproximadamente 500KBP (milhares de pares), speedups próximos do linear são obtidos com até 64 processadores e que a divisão do trabalho/comunicação tem pouca influência nos speedups. Para 128 processadores, no entanto, verificou-se que a divisão criteriosa de trabalho pode aumentar o speedup obtido de 100,41 para 122,75.**

I. Introdução

Na última década, os projetos genoma produziram uma enorme quantidade de dados referentes a sequências biológicas. Sempre que um novo organismo é sequenciado, sua sequência biológica é comparada com milhares de sequências de organismos já catalogados em bases de dados genômicas. Através dessa comparação, identificam-se similaridades que permitem que sejam feitas inferências sobre as características funcionais ou estruturais do organismo recém-sequenciado. Sendo assim, a comparação de sequências biológicas (ou alinhamento de sequências) é uma das mais importantes e fundamentais operações da biologia computacional.

O alinhamento de sequências consiste em determinar a melhor disposição de uma sequência sobre a outra introduzindo, quando necessário, espaços (*gaps*) nas mesmas, de modo a ressaltar as similaridades [1]. Os tipos mais comuns de alinhamento são local ou global. O objetivo de um algoritmo global é determinar a similaridade entre as duas sequências como um todo. Por outro lado, a comparação local é geralmente utilizada para determinar a similaridade entre partes das sequências.

O primeiro algoritmo exato para comparar duas sequências biológicas com custo fixo de *gap* foi o NW [2]. Este algoritmo possui complexidade quadrática de tempo e espaço, é baseado em programação dinâmica e calcula uma matriz de similaridade de tamanho $m \times n$, em que m e n são os tamanhos das sequências. O algoritmo NW foi modificado para tratar alinhamentos locais por Smith-Waterman (SW) [3]. Gotoh [4] modificou o algoritmo SW para o tratamento de *gaps* através de uma função mais biologicamente significativa (*affine gap*).

De maneira a reduzir o tempo de execução do algoritmo SW, foram propostos métodos heurísticos como o BLAST [5]. Estes métodos combinam filtros exatos com programação dinâmica para produzirem resultados mais rápido. A desvantagem dos métodos heurísticos é que a qualidade dos resultados produzidos (apesar de boa) é inferior à qualidade dos resultados obtidos pelos métodos exatos.

A computação de alto desempenho pode ser usada como uma alternativa para reduzir o tempo de execução dos métodos exatos. Como existe dependência de dados, a maioria das estratégias paralelas [6]–[9] utiliza o método da frente-de-onda [10] para calcular a matriz de similaridade. Em outros trabalhos [8], [9], foi observado que o tamanho do bloco possui influência no tempo total de execução do algoritmo e, por isso, deve ser configurado de maneira apropriada.

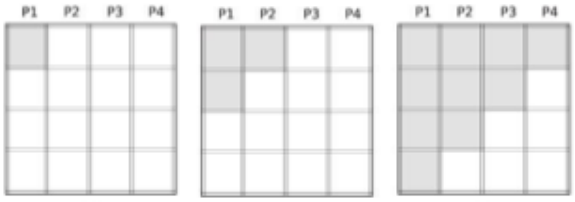
A maioria das abordagens paralelas presentes na literatura não apresenta estudos de speedup envolvendo mais de 100 máquinas para sequências longas, devido ao alto tempo necessário para essa operação. No entanto, acreditamos que tais estudos sejam necessários para a melhor compreensão do comportamento de tais algoritmos na presença de um grande número de máquinas.

Neste artigo, propomos e avaliamos uma variante paralela exata para execução do algoritmo SW em clusters compostos por mais de 100 máquinas. A nossa estratégia permite que o número de linhas da matriz de similaridade a serem processadas por cada processador seja parametrizado, permitindo que seja feito uma adaptação considerando o compromisso entre comunicação e computação.

Os resultados obtidos com 64 máquinas dispostas em 2 laboratórios da Universidade de Brasília, cada uma com 2 processadores, comparando sequências reais de DNA com tamanhos de 500KBP (milhares de pares) mostram que speedups muito próximos do linear são obtidos com até 64 processadores. Com até 64 processadores, os speedups parecem não ser influenciados pelo número de linhas atribuído a cada

		C	G	G	C	C	A	G	T	A
	C	1	0	0	1	1	0	0	0	0
	G	0	2	1	0	0	1	0	0	0
	G	0	1	3	0	0	1	0	0	0
	C	1	0	0	4	1	0	0	0	0
	C	1	0	0	1	5	0	0	0	0
	T	0	0	0	0	0	3	0	1	0
	A	0	0	0	0	0	1	1	0	2
	G	0	1	1	0	0	0	2	0	0
	T	0	0	0	0	0	0	0	3	0
	A	0	0	0	0	0	1	0	0	4

Tabela I: Matriz de similaridade e alinhamento local obtido



(a) (b) (c)

Figura 1: Método de frentes de onda

processador. Para 128 processadores, speedups muito bons são obtidos (de 100,41 a 122,75) e, nesse caso, observou-se que o desempenho é sensível ao número de linhas da matriz atribuído a cada processador. Com 128 processadores, o melhor speedup foi obtido quando cada processador processava 8 linhas em cada etapa.

O restante desse artigo está organizado como se segue. Na Seção II, é apresentado o problema de comparação de sequências biológicas, apresentando diversas estratégias para comparação paralela de sequências biológicas com. Na Seção III, a nossa estratégia paralela é proposta. A Seção IV apresenta e discute os resultados experimentais. Finalmente, a Seção V apresenta as conclusões e discorre sobre trabalhos futuros.

II. Comparação de Sequências Biológicas

As sequências de DNA podem ser tratadas como cadeias de caracteres compostas pelos elementos do alfabeto $\sum = \{A, T, G, C\}$. Como duas sequências de DNA são raramente idênticas, a comparação destas sequências é na verdade um problema de reconhecimento aproximado de padrões [1].

A. Escore de similaridade

Para que duas sequências biológicas sejam comparadas, deve-se achar o melhor alinhamento entre as mesmas. Isso é feito colocando-se uma sequência sobre a outra, tornando, assim, clara a correspondência entre caracteres similares [11].

O escore do alinhamento é obtido somando-se os valores atribuídos a cada coluna [12]. Na Tabela I, apresentamos um alinhamento entre duas sequências, na qual a pontuação de cada coluna é calculada da seguinte maneira: +1, se os caracteres são iguais (*match*), -2 se os caracteres são diferentes (*mismatch*) e -5 se ocorre um espaço (*gap*). Para esse alinhamento e esses valores de pontuação, o escore obtido é +4. A similaridade entre duas sequências é definida como sendo o maior escore [11].

B. Algoritmo SW para alinhamento local

O algoritmo SW [3] é um algoritmo exato baseado em programação dinâmica que obtém o melhor escore (similaridade) entre duas sequências biológicas em tempo e espaço quadrático. É executado em duas fases: criação da matriz de similaridade e obtenção do melhor alinhamento local.

1) Criação da matriz de similaridade: Esta fase recebe como entrada as sequências s e t, nas quais $|s| = m$ e $|t| = n$. A notação utilizada para representar o n-ésimo caractere de uma sequência u é $u[n]$ e, para representar um prefixo com n caracteres, usa-se $u[1 \ldots n]$. A matriz de similaridade é denotada por $D_{m \times n}$, em que $D_{i,j}$ contem o escore de similaridade entre os prefixos $s[1 \ldots i]$ e $t[1 \ldots j]$.

No início do processamento, as primeiras linha e coluna são preenchidas com zero. Os elementos restantes são obtidos através da Equação 1. Nessa equação, $p(i, j)$ é igual ao escore de um *match* se $s[i] = t[j]$, ou ao escore de um *mismatch* caso contrário e g é o escore de um *gap*. O valor de similaridade entre as sequências s e t é o maior valor presente na matriz. Sua posição marca o final do alinhamento local ótimo.

$$D_{i,j} = \max \begin{cases} D_{i,j-1} + g; \\ D_{i-1,j} + g; \\ D_{i-1,j-1} + p(i,j); \\ 0. \end{cases} \quad (1)$$

C. Variações paralelas do SW

No algoritmo descrito na Seção II-B, muito tempo é gasto no cálculo da matriz D e, por essa razão, esta é a parte que normalmente é paralelizada. O padrão de dependência de dados apresentado no cálculo da matriz é não-uniforme e sua estratégia de paralelização usa geralmente o método da frente de onda [10], já que os cálculos que podem ser feitos em paralelo evoluem como ondas sobre as diagonais da matriz.

A Figura 1 ilustra o método da frente de onda para 4 processadores. No início, somente P_1 pode calcular (Figura 1a). Quando P_1 termina de calcular os valores de uma coluna limítrofe, ele envia os mesmos para seu vizinho P_2, que pode começar o cálculo (Figura 1b). Na Figura 1c, o paralelismo máximo é atingido. Após isso, o paralelismo decresce novamente. Diversos trabalhos que propõem a paralelização do algoritmo SW têm sido publicados na literatura.

Em [6] é apresentada uma estratégia paralela *multithreaded* para comparação de sequências longas de DNA. O melhor speedup obtido foi 41, quando 64 processadores que compunham um cluster único foram utilizados para comparar duas sequências reais de 816.394BP e 580.074BP, respectivamente.

Em [7], é apresentada uma estratégia paralela com balanceamento de carga dinâmico para executar o algoritmo SW em múltiplos clusters. Uma estratégia hierárquica é utilizada,

onde cada cluster possui um nodo de controle. A comunicação inter-clusters usa mpich-g2 e a comunicação intra-cluster usa mpich. Além disso, o escalonamento de tarefas intra-cluster é controlado pelo escalonador *Sun Grid Engine* (SGE). Na comparação de duas sequências reais de DNA (144.260BP e 132.290BP) em 3 clusters de 8 processadores, dispostos em 3 laboratórios distintos, um speedup máximo de 13,2 foi alcançado quando 18 processadores eram utilizados.

Em [8] é proposta uma estratégia paralela com uso restrito de memória para a execução do SW e ajuste manual de quantidade de trabalho. O melhor speedup obtido com um cluster de 64 processadores (32 nodos dual core) foi de 33,64 quando duas sequências reais de DNA de tamanhos 3.147.090BP e 3.282.708BP foram comparadas.

Uma variante paralela ótima em espaço e tempo do SW foi proposta em [13]. O melhor speedup obtido com um cluster IBM Xseries de 60 processadores foi de 56, para o caso de *mismatch* total (em que todos os caracteres das sequências são diferentes). Nesse caso, sequências sintéticas de 320.000BP foram utilizadas.

O algoritmo FastLSA e sua versão paralela ParallelFastLSA são propostos em [14]. Esses algoritmos obtém o alinhamento global ótimo em espaço linear, reduzindo o tempo de execução, quando comparados com o algoritmo de Hirshberg [15]. A versão paralela foi executada em uma máquina SGI Origin 2400, com memória compartilhada entre 64 processadores. Enquanto todas as estratégias discutidas nos parágrafos anteriores utilizavam *Message Passing Interface* (MPI), o ParallelFastLSA usa threads em sua implementação. O melhor speedup (17,21) foi obtido na comparação de duas sequências reais de 319.030BP e 305.636BP, respectivamente.

Em [9], é proposto o uso do modelo CGM/BSP para uma versão paralela do SW. Um estudo detalhado sobre os impactos do tamanho do bloco na relação entre comunicação e computação é feito. O algoritmo foi implementado em MPI e o melhor speedup (39,9) foi obtido com 64 processadores na comparação de sequências de 8KBP $\times$ 16KBP. O speedup de 39,9 foi inferido a partir da execução com 4 processadores.

A Tabela II apresenta uma visão comparativa de variantes paralelas do algoritmos SW propostas na literatura. Como pode ser visto, a maioria dos algoritmos [6]–[8], [13] foi programado com MPI. Somente [14] usou memória compartilhada com threads. Com exceção de [13], os speedups obtidos com as maiores sequências não chegou perto do speedup linear, o que indica que essas abordagens pode sofrer de problemas de escalabilidade. Em [13] foi obtido um speedup de 56 em 60 processadores porém as sequências comparadas eram sintéticas e inteiramente diferentes. Não está claro que um speedup dessa ordem se manterá na comparação de sequências reais.

A estratégia proposta no presente artigo atingiu um speedup de 121,75 comparando sequências reais de aproximadamente 500KBP com 128 processadores, distribuídos em 2 laboratórios, interligados por uma rede ethernet. A nosso conhecimento, essa é a primeira vez que um estudo de speedup de variante paralela do SW é feito para clusters não confinados de mais de 64 processadores.

Ref.	Proc.	Speedup	cluster	Tam. comp. (BP)
[7]	18	13,20	heterogêneo não confinado MPI	144.260 $\times$ 132.290
[14]	32	17,20	homogênio confinado threads	319.030 $\times$ 305.636
[8]	64	33,64	homogênio confinado MPI	3.147.090 $\times$ 3.282.708
[9]	64	39,90	homogênio confinado MPI	8.192 $\times$ 16.384
[6]	64	41	homogênio confinado MPI	816.394 $\times$ 580.074
[13]	60	56	homogênio confinado MPI	320.000 $\times$ 320.000
nossa	128	122,75	homogênio múltiplos labs. MPI	1.044.459 $\times$ 1.072.950

Tabela II: Características das estratégicas paralelas

Figura 2: Divisão do trabalho entre 4 processadores

III. Projeto da Estratégia Paralela Escalável

A estratégia paralela proposta nessa seção calcula o escore local ótimo com P processadores em espaço linear. Ao final dessa fase, o escore de similaridade é fornecido como saída.

Dadas duas sequências s e t, de tamanhos n e m, respectivamente, em que m é o tamanho da menor sequência, cada processador irá calcular um subconjunto de m/p colunas da matriz de similaridades. Logo, cada processador necessita da maior sequência inteira e de parte da menor sequência. A Figura 2 ilustra a divisão do trabalho entre 4 processadores.

Ambas as sequências estão em arquivos gerenciados pelo *Network File System* (NFS) e os processadores lêem esses arquivos.

O processador P_1 começa o processamento, enquanto os outros processadores ficam bloqueados até que uma mensagem seja enviada. Ao terminar o cálculo de um bloco contendo k linhas, P_1 envia os elementos da borda para P_2, que começa

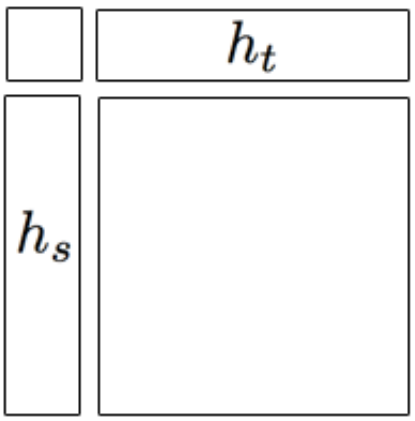


Figura 3: Informação necessária para o processamento de cada bloco da matriz

Algoritmo 1: Pseudo-código do algoritmo utilizado na comunicação entre os processadores

```
inicialização;
alocação de memória;
ler toda a segunda sequência do arquivo;
for blocos de processamento do
    ler um bloco da primeira sequência do arquivo;
    if rank do processador > 0 then
        receber do processador anterior o bloco h_s;
    processar os dados do bloco;
    if rank do processador < p − 1 then
        enviar ao próximo processador o bloco h_s;
if n mod p > 0 then
    ler um bloco da primeira sequência do arquivo;
    if rank do processador > 0 then
        receber do processador anterior o bloco h_s;
    processar os dados do bloco;
    if rank do processador < p − 1 then
        enviar ao próximo processador o bloco h_s;
```

a processar o seu primeiro bloco. Nesse momento, P_1 e P_2 estão processando em paralelo. O mesmo processo se repete entre cada dois processadores vizinhos até que a frente de onda esteja cheia, ou seja, todos os processadores estejam processando.

O Algoritmo 1 apresenta os passos executados na nossa estratégia paralela. A alocação de memória em cada processador é feita da seguinte maneira. Seja k o número de linhas calculadas em cada iteração. Logo, são armazenados os elementos h_s e h_t, dois buffers de tamanho k e n/p, respectivamente, utilizados para enviar e receber dados entre os processadores. Como o último processador é responsável por processar os elementos restantes da sequência n, o seu buffer h_t tem tamanho $n/p + n \mod p$. Além disso, cada processador deve alocar espaço para a matriz $D_{m \times n}$. Cada elemento utiliza 4 ou 8 bytes, dependendo do tamanho da maior sequência.

Tam. aprox.	Tam. real	Nome
500KBP	542869BP	*Agrobacterium tumefaciens*
500KBP	563165BP	*Rhizobium sp.*
1MBP	1044459BP	*Chlamydia trachomatis*
1MBP	1072950BP	*Chlamydia muridarum*

Tabela III: Organismos comparados

IV. Resultados Experimentais

A variante paralela do SW que implementa a estratégia proposta na Seção III foi programada em C e OpenMPI. Nossos testes foram feitos em dois laboratórios do Departamento de Ciência da Computação da Universidade de Brasília (UnB), onde cada laboratório tinha 32 máquinas Intel Core 2 Duo 2.93GHz, 2 GB RAM, totalizando 128 processadores. Os laboratórios estavam conectados por um *switch* de 100Mbps e os testes foram realizados em ambiente dedicado. Todos os cores executam o Linux Debian Lenny, kernel 2.6.26-2-686 i686, e OpenMPI 1.2.7.

Foram usadas sequências reais de DNA obtidas do site do *National Center for Biotechnology Information* (NCBI). O número de linhas da matriz processadas por cada core foi 2, 4, 8 e 16. Esses valores foram escolhidos de maneira empírica, pois desejávamos ver o impacto do tamanho do bloco de processamento no tempo total de execução.

O tempo total de execução e os escores obtidos são apresentados na Tabela IV. Para os dois casos, a nossa estratégia foi executada com 1, 2, 4, 8, 16, 32, 64 e 128 processadores. Na execução com 1 processador, foi utilizado o programa paralelo, configurado para utilizar apenas 1 processador.

O programa ssearch, disponível publicamente na suite do pacote FASTA, alinhou as mesmas sequências em 62792s, ou seja, um pouco mais do que 17 horas. Como o ssearch utiliza o modelo affine gap e, por isso, calcula duas matrizes adicionais com o mesmo *wavefront*, considera-se que o SW puro demora um terço do tempo do SW com affine gap. Isso nos dá cerca de 5 horas, o que mostra que o tempo sequencial do nosso programa é comparável com o tempo de um programa popular de cálculo de SW.

Para alinhar sequências de 500KBP com 1 processador, foram necessárias um pouco mais de 5 horas (18176s). Com 128 processadores, no entanto, o mesmo resultado foi obtido em menos de 3 minutos (151s).

Os speedups obtidos na comparação entre as sequências de 500KBP, com 1, 2, 4, 8, 16, 32, 64 e 128 processadores, utilizando 2, 4, 8 e 16 linhas por bloco de processamento são mostrados na Figura 4. Como pode ser visto, os speedups são muito parecidos até 64 processadores e parecem não depender do número de linhas por bloco. No entanto, com 128 processadores, o melhor speedup (120,37) é obtido com 8 linhas por bloco. Os outros speedups estão bem próximos, na faixa que vai de 95,16 a 102,69. Isso mostra que, para esse caso, o número de linhas por bloco é um parâmetro importante e que tal diferenciação não estaria clara se calculássemos o speedup somente até 64 processadores.

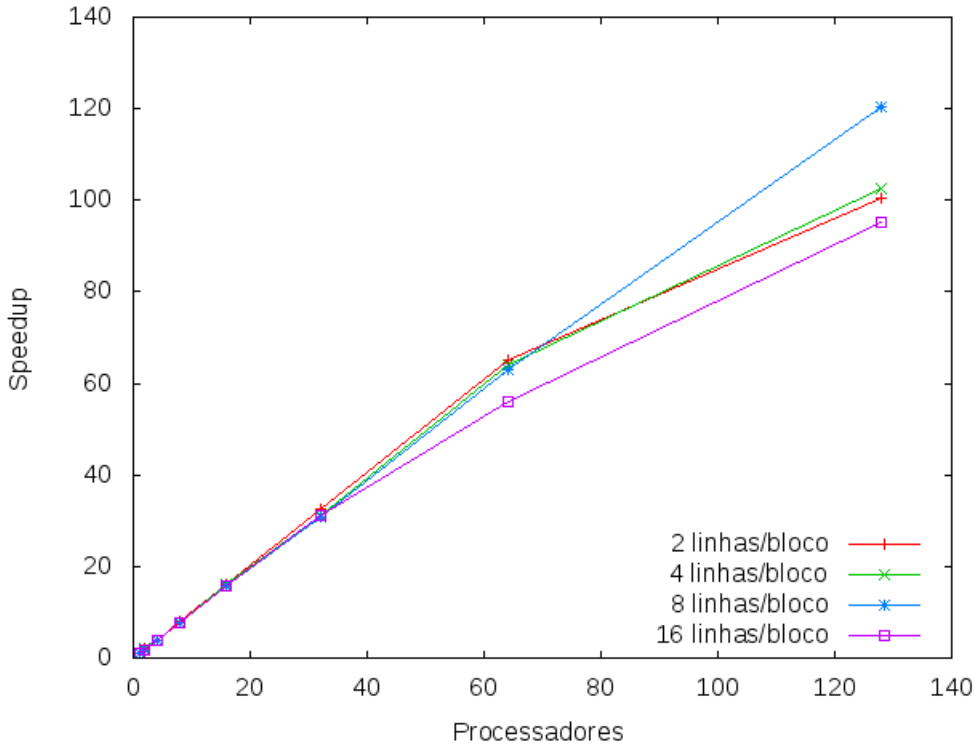


Figura 4: Speedups obtidos para as sequências de 500KBP

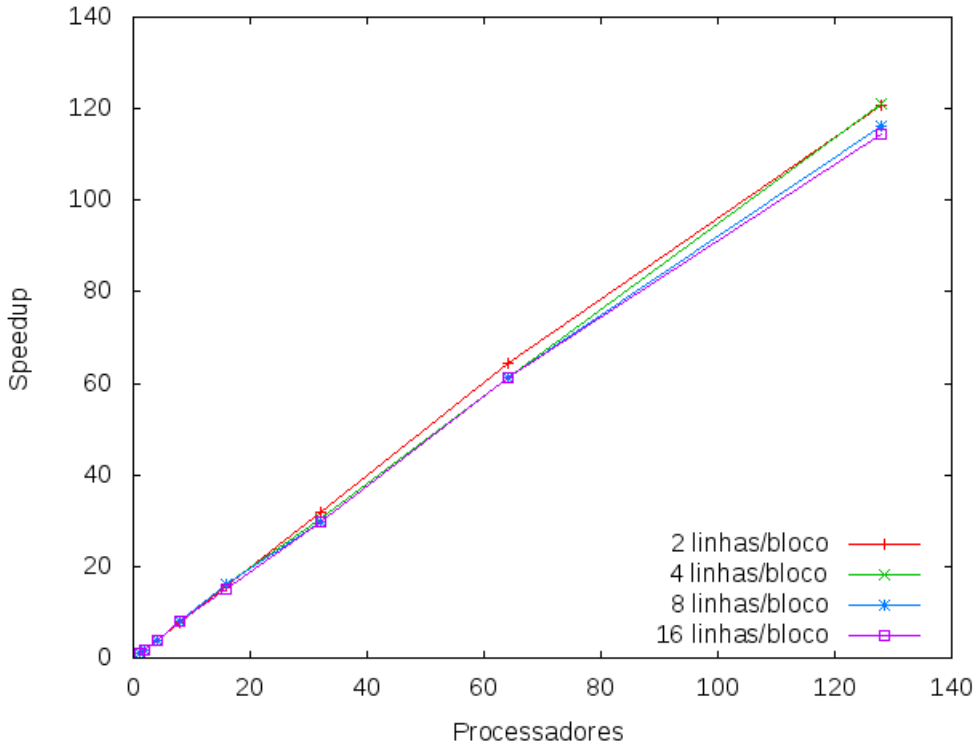


Figura 5: Speedups obtidos para as sequências de 1MBP

Os speedups obtidos na comparação entre as sequências de 1MBP, com 1, 2, 4, 8, 16, 32, 64 e 128 processadores e 2, 4, 8 e 16 linhas por bloco de processamento são mostrados na Figura 5. Nesse caso, os melhores speedups são obtidos com

Tam. seq.	1	2	4	8
500KBP	18176s	9424s/2 9227s/4 9520s/8 9520s/16	4557s/2 4648s/4 4630s/8 4652s/16	2247s/2 2311s/4 2317s/8 2324s/16
	16	**32**	**64**	**128**
500KBP	1119s/2 1137s/4 1162s/8 1165s/16	561s/2 584s/4 589s/8 586s/16	279s/2 284s/4 289s/8 325s/16	181s/2 177s/4 151s/8 191s/16
	1	**2**	**4**	**8**
1MBP	42907s	23057s/2 22809s/4 22896s/8 22948s/16	11326s/2 11467s/4 11492s/8 11506s/16	5638s/2 2863s/4 2863s/8 2869s/16
	16	**32**	**64**	**128**
1MBP	2754s/2 2863s/4 2863s/8 2869s/16	1343s/2 1402s/4 1443s/8 1442s/16	667s/2 699s/4 699s/8 699s/16	355s/2 354s/4 369s/8 375s/16

Tabela IV: Tempos de execução/k para as sequências

2 e 4 linhas por bloco (128,50 e 128,86, respectivamente). O pior speedup foi de 121,64, o que pode ser considerado muito bom, para o caso de 128 processadores.

V. Conclusões e Trabalhos Futuros

Neste artigo, propusemos e avaliamos uma estratégia paralela para execução do algoritmo SW com MPI em múltiplos laboratórios de pesquisa, interconectados por rede local.

Os resultados obtidos em 2 laboratórios de pesquisa, com um total de 128 processadores, apresentaram speedups muito bons para comparações entre sequências de 500KBP e 1MBP. O tempo total de execução para alinhar sequências de 1MBP com 128 processadores foi de 5 minutos e 54 segundos. A execução com 2 processadores demorou 6 horas, 19 minutos e 48 segundos, o que nos dá uma redução impressionante do tempo de execução.

Como trabalhos futuros, pretendemos testar nossa estratégia com um número maior de processadores e também implementá-la em GPU, fazendo a avaliação comparativa entre as duas abordagens (cluster e GPU).

Referências

[1] G. Navarro, "A guided tour to approximate string matching," *ACM computing surveys (CSUR)*, vol. 33, no. 1, p. 88, 2001.

[2] S. Needleman and C. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[3] T. Smith and M. Waterman, "Identification of common molecular subsequences," *J. Mol. Bwl*, vol. 147, pp. 195–197, 1981.

[4] O. Gotoh, "An improved algorithm for matching biological sequences," *Journal of Molecular Biology*, vol. 162, no. 3, pp. 705–708, 1982.

[5] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic local alignment search tool," *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.

[6] W. Martins, J. del Cuvillo, W. Cui, and G. Gao, "Whole genome alignment using a multithreaded parallel implementation," in *Symposium on Computer Architecture and High Performance Computing*. Citeseer, 2001, pp. 1–8.

[7] C. Chen and B. Schmidt, "An adaptive grid implementation of DNA sequence alignment," *Future Generation Computer Systems*, vol. 21, no. 7, pp. 988–1003, 2005.

[8] A. Boukerche, R. B. Batista, A. C. Melo, F. B. Scarel, and L. A. B. Souza, "Exact parallel alignment of megabase genomic sequences with tunable work distribution," *International Journal of Foundations of Computer Science*, 2010.

[9] C. Alves, E. Cáceres, F. Dehne, and S. Song, "A parallel wavefront algorithm for efficient biological sequence comparison," *Computational Science and Its Applications—ICCSA 2003*, pp. 973–973, 2003.

[10] G. Pfister, *In search of clusters: the coming battle in lowly parallel computing*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1995.

[11] J. Setubal and J. Meidanis, *Introduction to computational molecular biology*. Pws Publishing, 1997.

[12] D. Gusfield, *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge Univ Pr, 1997.

[13] S. Rajko and S. Aluru, "Space and time optimal parallel sequence alignments," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1070–1081, 2004.

[14] A. Driga, P. Lu, J. Schaeffer, D. Szafron, K. Charter, and I. Parsons, "Fastlsa: a fast, linear-space, parallel and sequential algorithm for sequence alignment," *Algorithmica*, vol. 45, no. 3, pp. 337–375, 2006.

[15] D. Hirschberg, "A linear space algorithm for computing maximal common subsequences," *Communications of the ACM*, vol. 18, no. 6, p. 343, 1975.