Aplicação de processamento paralelo no problema de reconhecimento de genes

Leonardo M. Takuno
Líria Matsumoto Sato
Edson T. Midorikawa

Universidade de São Paulo, Escola Politécnica da Universidade de São Paulo,
PCS - Depto. Engenharia de Computação e Sistemas Digitais,
Av. Prof. Luciano Gualberto, travessa 3, no. 158
CEP:05508-900, São Paulo, SP
Tel : 55-11-3091-5617
{leonardo.takuno,liria.sato,edson.midorikawa}@poli.usp.br

Resumo

Nos últimos anos, a quantidade de dados biológicos em bancos de dados públicos vem aumentando exponencialmente. Como conseqüência, os algoritmos de análises de dados, em especial algoritmos de predição de genes, estão associados a longos tempos de execução. Neste sentido, a computação paralela é um meio eficaz para diminuir o tempo de execução do processamento de tais análises.
Neste artigo, apresentaremos uma solução paralela para o tratamento do problema de reconhecimento de genes, problema este de grande relevância biológica e computacional. Para isso utilizaremos um algoritmo de predição de genes conhecido por Sim4. Este algoritmo foi projetado para alinhar seqüências de DNA complementar (cDNA) com seqüências de DNA genômico, e produzir como resultado uma tabela contendo as localizações das regiões que codificam proteínas.

Palavras chaves: computação paralela, Sim4, predição de genes, distribuição de dados, aglomerados de computadores.

1. Introdução

O desenvolvimento de tecnologia de seqüenciamento de DNA em larga escala proporciona uma grande quantidade de novos trechos de DNA não caracterizados, os quais são dados rudimentares que precisam ser analisados para descobrir a estrutura e função do DNA seqüenciado.

A primeira fase da análise, também conhecida como anotação, consiste em interpretar os dados biológicos e tratar do problema conhecido como **problema de reconhecimento de genes**, que segundo Guigó em [5] é o problema de deduzir seqüências de aminoácidos em uma dada seqüência de DNA, isto é, deseja-se encontrar regiões de DNA que codificam proteínas. Nos últimos quinze anos, este problema, de grande relevância biológica e computacional, tem recebido a atenção dos pesquisadores e um grande esforço tem sido dispendido na busca de soluções.

Entretanto, segundo Mathé *et. al*. em [12], a tarefa de encontrar genes em uma seqüência genômica não é um problema trivial. Vários métodos foram propostos e são classificados em dois grupos: a abordagem intrínseca e abordagem extrínseca. A primeira abordagem utiliza métodos estatísticos, modelos estocásticos, redes neurais, entre outras técnicas, para identificar certas informações contidas na própria seqüência de DNA. Alguns exemplos de programas que utilizam método intrínseco incluem: FGENEH [18], GENMARK[2], GeneID [6], Genie [11]. A Figura 1 apresenta a estrutura de um gene eucarionte e as informações contidas nesta estrutura.

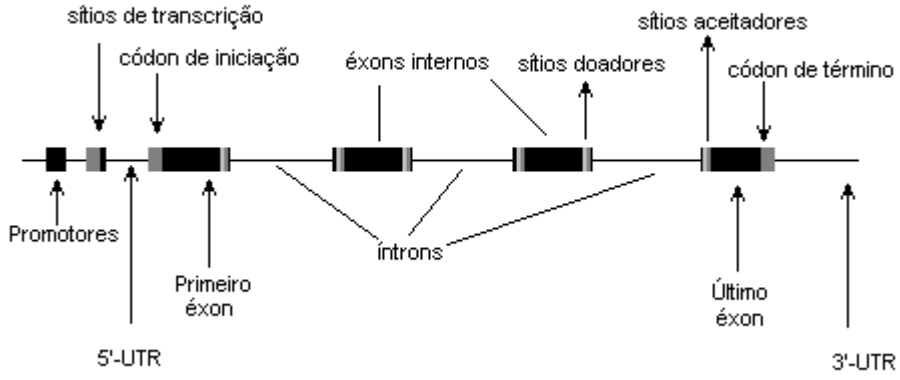

Figura 1: Estrutura de um gene eucarionte.

A segunda abordagem compara a seqüência a ser anotada com informações contidas em seqüências previamente anotadas, as quais são seqüências de DNA de outros organismos, etiquetas de seqüências expressas (EST), DNA complementar (cDNA), RNA mensageiro (mRNA) e proteínas. Geralmente, esta comparação é feita através de algoritmos de alinhamento que computam a similaridade entre duas seqüências, os quais são baseados em programação dinâmica proposta por Needleman e Wunsch em [15] e Smith e Waterman em [17]. Porém, estes algoritmos executam em tempo proporcional a O(MN) e requerem espaço O(MN), onde M é o tamanho da seqüência a ser anotada e N é o tamanho da seqüência previamente anotada. Procrustes [7] e EST_GENOME[14] são exemplos de algoritmos baseados em programação dinâmica e levam em consideração a estrutura exon-intron dos organismos eucariontes. Visto que estes algoritmos requerem grande quantidade de tempo de processamento e de espaço de memória, é impraticável a aplicação destes métodos em grande volume de dados. Por outro lado, heurísticas, tais como o FASTA[16] e BLAST[1], são métodos mais rápidos para computar o alinhamento entre seqüências e são amplamente utilizados por pesquisadores. Dentre os algoritmos de reconhecimentos de genes baseados em homologia que utilizam heurísticas baseadas no BLAST estão: Sim4[4], BLAT[10] e Spidey[19].

Contudo, o processo de anotação não está acompanhando a avalanche de dados, que diariamente são disponibilizados devido a diversos projetos genomas existentes. Este grande volume faz com que os atuais algoritmos de reconhecimento de genes estejam associados a longos tempos de execução. Isto requer a criação de ferramentas mais rápidas no tratamento destes dados.

Uma solução muito utilizada neste caso é o uso de sistemas computacionais de alto desempenho, como os aglomerados (*cluster*) de computadores, os quais são sistemas de baixo custo e escaláveis.

Assim, a aplicação do processamento paralelo pode reduzir substancialmente o custo e o tempo exigido para caracterizar tais dados.

Neste artigo, nós pretendemos explorar o processamento paralelo em um *cluster* de computadores, propor, implementar e analisar uma solução paralela no tratamento do problema de reconhecimento de genes. Para isso, utilizaremos o algoritmo Sim4 [4] como componente, o qual foi escolhido devido à velocidade de execução no algoritmo seqüencial, e pela sua popularidade, sendo este algoritmo muito utilizado em pesquisas biológicas.

O artigo está organizado em 6 seções. A seção 2 apresenta uma breve descrição da síntese de proteínas. A seção 3 apresenta a descrição do algoritmo Sim4. A seção 4 apresenta abordagens paralelas para o tratamento do problema de reconhecimento de genes. A seção 5 apresenta os casos de testes realizados. A seção 6 apresenta a conclusão.

2. Síntese de proteínas

Para entender o problema de identificação de genes é importante descrevermos o mecanismo biológico de síntese de proteínas. A primeira fase da síntese de proteínas é conhecida como a fase de **transcrição**, quando é produzida uma cópia de RNA a partir de uma das fitas do DNA. Para os organismos procariontes, aqueles desprovidos de membrana nuclear (no caso de bactérias e algas azuis), este RNA é chamado RNA mensageiro (mRNA), que representa exatamente a seqüência genômica, exceto pelo fato de ter uracil (U) onde havia timina (T) no gene [13]. Para os organismos eucariontes, os quais possuem núcleo separado do resto da célula por uma membrana nuclear, a fase de transcrição é um pouco mais complexa. Neste caso, o transcrito imediato do gene é um pré-mRNA que deve ser processado para gerar um mRNA maduro.

O estágio mais importante do processamento é o ***splicing***[1] **do RNA**. Muitos genes em eucariontes contêm regiões internas que codificam proteínas, chamados éxons, e são interrompidos por regiões que não participam desse processo, os íntrons. Os limites entre éxons e íntrons são denominados sítios de *splicing*. O processo de *splicing* remove os introns do pré-mRNA e dão origem ao mRNA, que consiste de uma série de éxons unidos na mesma ordem em que estão no DNA.

Neste artigo, trataremos somente de genes eucarióticos, cujo mecanismo de síntese de proteína é ilustrado na Figura 2.

Devido a este mecanismo, diferentes nomes são utilizados para referenciar o gene completo e o gene sem íntrons. O primeiro é chamado DNA genômico e o segundo de DNA complementar (cDNA). O cDNA pode ser obtido a partir do mRNA através do processo chamado **transcrição reversa.**

Após a transcrição, o mRNA produzido será usado na fase conhecida como **tradução**. Nesta fase, cada tripla de bases nitrogenadas, conhecidas por códons, é traduzida, com o auxílio do RNA-transferidor e o RNA-ribossômico, para aminoácidos constituintes da proteína a ser sintetizada.

Gelfand *et. al.* [8] trata o problema de reconhecimento de genes utilizando uma abordagem combinatória para encontrar um conjunto de blocos em seqüência genômica, cuja concatenação

[1] *Splicing* é um termo usado internacionalmente que pode ser traduzido, em português, como ‘cortar’ e ‘ligar’.

melhor se ajuste em uma seqüência alvo. Este problema é conhecido como o **problema de alinhamento *spliced***, e explora todas as montagens possíveis de blocos e encontram uma montagem com mais alta pontuação. Após a montagem dos blocos, espera-se que essa montagem represente a estrutura exon-intron.

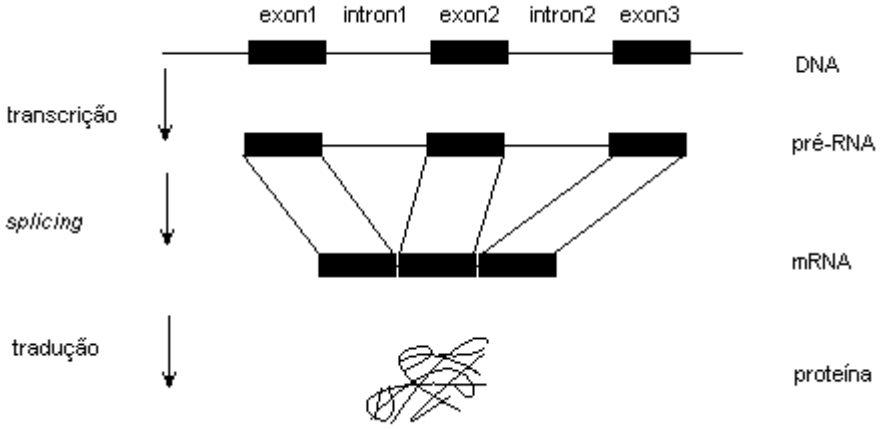

Figura 2: Síntese de proteínas.

.

A definição do problema de alinhamento *spliced* é descrita a seguir:

Seja G = $g_1 \ldots g_n$ uma string de letras e sejam B = $g_i \ldots g_j$ e B' = $g_i' \ldots g_j'$ substrings de G. Denota-se B < B' se j < i', isto é, se B termina antes do início de B'. Uma seqüência $\Gamma = (B_1, \ldots, B_p)$ de substrings de G é um encadeamento se $B_1 < B_2 < \ldots < B_p$.

Denota-se a concatenação de strings do encadeamento Γ por $\Gamma^* = B_1 * B_2 * \ldots * B_p$. Dadas duas strings G e T, s(G, T) denota a pontuação do alinhamento ótimo entre G e T.

Seja G = $g_1 \ldots g_n$ uma string chamada seqüência genômica, T = $t_1 \ldots t_m$ uma string chamada seqüência alvo e $B = \{ B_1, \ldots, B_b \}$ um conjunto de substrings de G que representam os blocos. Dado G, T e *B*, o problema do alinhamento *spliced* consiste em encontrar o encadeamento Γ de strings de *B* tal que a pontuação $s(\Gamma^*,T)$ do alinhamento entre a concatenação de string e da seqüência alvo é máxima entre todos os encadeamentos de blocos de *B*.

Esta definição é adequada para este problema, entretanto como estamos utilizando uma heurística baseada no BLAST para determinar tais alinhamentos, a pontuação $s(\Gamma^*,T)$ pode não ser um alinhamento ótimo, mas pode ser bom o suficiente que seja aceitável para os propósitos biológicos.

3. O algoritmo Sim4

O Sim4 (Florea *et. al.* 1998)[4] é um algoritmo de alinhamento *spliced* desenvolvida com o objetivo de alinhar seqüências de cDNAs com seqüências genômicas e foi utilizado no projeto genoma BDGP (*Berkeley Drosophila Genome Project*) para anotação dos genes provenientes da mosca da fruta, conhecida como *Drosophila melanogaster*. Inicialmente, esta tarefa era realizada utilizando-se uma ferramenta para busca de similaridade conhecida como BLAST (*Basic Local Alignment Search Tool*) [1], a qual nem sempre apresentava resultados exatos. Assim, os biólogos utilizavam certa intervenção manual para identificar as possíveis regiões codificantes. Mas o considerável crescimento do número de seqüências produzidas levou à necessidade de uma ferramenta mais eficiente para execução daqueles alinhamentos.

O Sim4 recebe dois arquivos como parâmetros de entrada, um deles contendo seqüências de cDNA e outro contendo seqüências genômicas. Estes arquivos contêm seqüências em formato conhecido como FASTA (veja a Figura 3 como exemplo), o qual inicia com uma linha de descrição, que contém o nome da seqüência e as linhas seguintes contêm a seqüência em si. A linha de descrição se distingue das linhas de seqüência por conter em seu início o sinal de maior (“>”).

```
>gi|42591982|gb|CK738468.1|CK738468 00210051083.C11_01110110VC.scf IR62266 Oryza
sativa cDNA clone 00210051083.C11_01110110VC.scf similar to unnamed protein product
[Homo sapiens], mRNA sequence
GGTCCGATCCGGCCGGCCGGCGGCGATGGCGCCCGTGGCGACGTGGTGCTGGTGCAGCCACTGCGGCGTG
GACCGTCGCCTGAGGAGGGAGGGCGACTACGCCTCCTGCAGCTCCTGCGGCAAGGTCCTCCTGCAGCTCC
GGGGTGACGACGACGCCGCCGCCGCCGAGGCCCCTCGCCTTCGCCTGCTCCTGGGGCCGAGGATGCGGAG
GACCAAAAAGCGCGCGGCGGCGGCTGATGCTGCAGGCGGCGGTGGAGTTGCCACCGGCGGCGGCCGCGGC
>gi|42583857|gb|CK730343.1|CK730343 d70 cDNA subtractive library of human rectum
adenocarcinoma Homo sapiens cDNA, mRNA sequence
ACGCGGAGGCTTTGGAGGGCGAGGAGGCTTCCGAGGAGGCAGAGGAGGAGGAGGTGACCACAAGCCACAA
GGAAAGAAGACGAAGTTTGAATAGCTTCTGTCCCTCTGCTTTCCCTTTTCCATTTGAAAGAAAGGACTCT
GGGGTTTTTACTGTTACCTGATCAATGACAGAGCCTTCTGAGGACATTCCAAGACAGTATACAGTCCTGT
CCCATGT
```

Figura 3: Exemplo de arquivo de entrada.

Cada arquivo pode conter uma coleção de seqüências em formato FASTA, neste caso, o termo “banco de dados” é bastante utilizado para denominar tais arquivos. Porém, estes são arquivos textos sem qualquer operação de Sistemas de Gerenciamento de Banco de Dados (SGBD).

Em linhas gerais, o algoritmo Sim4 realiza os seguintes passos:

1. Primeiramente, são encontrados todos os pares de segmentos (*segment pair*). Dadas duas seqüências, um **par de segmentos** é definido como um par de subseqüências de mesmo comprimento que formam um alinhamento sem buracos (*gaps*). Este algoritmo determina todos os pares de segmentos entre uma seqüência de cDNA com uma seqüência genômica. Após isso, o algoritmo determina os melhores pares de segmentos, conhecidos como **HSP** (*high-scoring segment pair*). Um HSP, na terminologia do BLAST[1], consiste de dois fragmentos de seqüência de tamanho arbitrário, mas de tamanho igual cujo alinhamento local é máximo, e que a pontuação (*score*) não excede a um valor limiar (*threshold*) ou uma pontuação de corte. A tarefa de encontrar os HSPs inicia com a identificação de palavras de tamanho fixo k, cujo valor *default* para k é 12. Para isso, o algoritmo utiliza uma tabela de espalhamento (*hash*). Estas palavras são estendidas para a esquerda e para a direita, pontuando com valor 1 as igualdades e penalizando com valor –5 para as desigualdades sem o uso de buracos. Esse prolongamento do alinhamento inicial termina quando a pontuação do alinhamento estendido começa a diminuir.
2. A seguir, um conjunto de HSPs com chances de representar um éxon é selecionado. Esta seleção pode ser feita por meio de um algoritmo de encadeamento unidimensional proposto por Gusfield em [7] e apresentado no Algoritmo 1 abaixo. Este algoritmo maximiza a pontuação (score) total de um subconjunto de intervalos, dois a dois disjuntos, de acordo com uma função de pontuação previamente conhecida.

Definição: Dado o intervalo $I \subset \{[a,b] \in R^2\}$, dizemos que I é independente se dois intervalos $i_1, i_2 \in I$ quaisquer são sempre disjuntos.

Algoritmo 1 : Encadeamento unidimensional.

```
Entrada: I ⊂ { [a,b] ∈ R² }     <conjunto de intervalos>
         w : I → R              <função de pontuação>
Saída: J ⊆ I independente tal que w(J) é máximo
Encadeamento ( I , w )
início
        k ← |I|
        Enumere os intervalos i ∈ I = {i_j} j = 1,2,..,k, onde i_j=(a_j,b_j)
        L ←{(a_j,j,'a')} j =1, ...,k ∪ {(b_j,j,'b')} j =1, ...,k
        Ordenar L = {(x, j, t)} de cada aresta com ordens crescente de x.
        P← 0   <P é o peso máximo já encontrado>
        J←{}   <J é subconjunto de I>
        para cada (x, j, t) ∈ L <selecione de forma ordenada>
                se t ='a' então <início do intervalo>
                        v[j] ← p + w(i_j)
                senão    <t = 'b' – fim do intervalo>
                        se p < v[j] então
                                p  ← v[j]
                                para cada I_l ∈ J
                                        se a_j<b_l então   <remove i_l de J>
                                                J←J\{i_l} <garante J independente>
                                        fim-se
                                fim-para
                                J←J ∪ {i_j} <adiciona i_j a J>
                        fim-se
                fim-se
        fim-para
        devolva( p , J )  < p = w(J) >
fim algoritmo
```

3. Quando possíveis éxons (*exon cores*) consecutivos, dados por uma coleção de HSPs muito próximos apresentam alguma sobreposição na seqüência expressa, suas posições finais são aparadas para formar um íntron entre GT e um AG (ou entre CT e AC)[2]. Quando não ocorre nenhuma sobreposição, os possíveis éxons são estendidos um em direção ao outro até que se encontrem. Este procedimento elimina os buracos existentes no alinhamento. Similarmente, o primeiro e último éxon possível são estendidos ao fim da seqüência de cDNA.
4. Identificados os limites de cada éxon, estes últimos são alinhados com a seqüência original por meio de um programa de alinhamento descrito por Chao *et. al.* [3].

A Figura 4 apresenta a saída do algoritmo Sim4 com as posições do alinhamento *spliced.*

```
seq1 = Homo40.fasta, 892 bp
seq2 = Homo100.fasta (gi|42627474|emb|BX927320.5|), 100409 bp

748-775  (52394-52420)   85% --
776-793  (54953-54970)   100% ==
814-832  (62574-62592)   100% ->
833-843  (66366-66376)   81% <-
844-859  (83257-83272)   93% ->
860-866  (92131-92137)   100% ->
867-884  (94927-94942)   88%
```

Figura 4: Saída do Sim4.

[2] GT e AG ou CT e AC são sítios de *splicing.*

4. Uma solução paralela para o reconhecimento de genes

Uma solução paralela eficiente depende da decomposição de trabalho entre os processadores disponíveis. Algoritmos de reconhecimento de genes, enfoque principal deste trabalho, envolve um grande processamento de dados, decorrentes das inúmeras comparações entre seqüências de cDNAs e seqüências genômicas, com a finalidade de obter o alinhamento *spliced*. O processamento paralelo das comparações entre seqüências, distribuídas aos vários processadores, reduziria o tempo de execução. O ganho de desempenho é fortemente dependente da estratégia adotada para a distribuição, tanto das seqüências de cDNAs como as genômicas. Apresentamos aqui uma solução paralela, onde foi desenvolvida uma estratégia de distribuição de seqüências.

Esta solução foi implementada em linguagem C e biblioteca de passagem de mensagem MPI (*Message Passing Interface*)[9]. Nesta aplicação existe um processo especial que controla a execução dos demais processos, denominado **processo mestre** ou simplesmente **mestre**. Os demais processos são denominados **escravos**. O acesso do banco de dados é feito através do sistema de arquivos de rede (NFS – *Network File System*), dispensando a transferência de dados pelos nós da rede. O modelo adotado é o paradigma de programação SPMD (*Single Program, Multiple Data*), no qual cada processo executa o mesmo programa. Apesar disso, os processos podem executar instruções diferentes, dependendo do seu identificador.

O programa recebe como parâmetros de entrada dois arquivos, ou bancos de dados, cada um contendo um conjunto de seqüências, tais como descrito a seguir: Sejam dois conjuntos de seqüências de entrada R e S. R é o conjunto de seqüências $\{r_1, r_2, \ldots, r_m\}$, onde $r_i \in$ R, e r_i é uma seqüência de cDNA, e m é o número de seqüências em R. S é um conjunto de seqüências $\{s_1, s_2, \ldots, s_n\}$, onde $s_j \in$ S, e s_j é uma seqüência genômica, e n é o número de seqüências em S.

4.1 Processamento paralelo de pares de seqüência

Nesta abordagem, o mestre é responsável pela distribuição de dados e pela união dos resultados. A atribuição de tarefas aos processos escravos ocorre de maneira dinâmica. Os pares de seqüências são processados em paralelo pelos processos escravos e serão passados como parâmetros de entrada do Sim4, sem dependência de dados.

Inicialmente, o mestre divide os conjuntos R e S em *m e n* arquivos temporários, respectivamente. Isto é feito por meio de um pré-processamento dos conjuntos. Cada arquivo armazena uma única seqüência a ser analisada. A seguir, são criadas duas listas, uma para armazenar os nomes dos arquivos gerados a partir de R e outra para armazenar os nomes dos arquivos gerados a partir de S. A comparação entre os dois conjuntos R e S, com *m* e *n* elementos respectivamente, resulta em O($m \times n$) alinhamentos a serem processados.

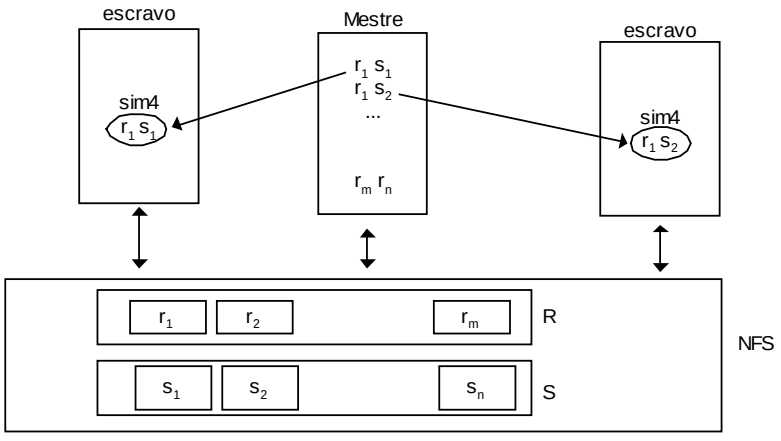

Figura 5: Processamento paralelo de pares de seqüência.

Ao iniciar o processamento, os nós escravos solicitam ao mestre uma tarefa a ser processada. A seguir, o mestre envia para os escravos solicitantes um par de nomes de arquivos temporários. Estes escravos recebem o par de nomes de seqüências e passam como parâmetros ao Sim4. Após o processamento, os resultados são gravados em um arquivo de saída. A seguir, o escravo envia solicitação de uma nova tarefa. E assim sucessivamente, até que não existam mais pares de seqüências a serem processadas. Enquanto alguns escravos encontram-se ainda processando seu par de seqüências devido ao tamanho dos arquivos, outros terminam o processamento e solicitam mais tarefas. Desta maneira, ocorre um balanceamento de carga natural, já que os tamanhos das seqüências de entrada são diferentes.

Nesta versão o mestre não participa do processamento dos pares de seqüências, limitando-se em atender as requisições dos escravos. A Figura 5 a seguir ilustra esta abordagem de paralelização do algoritmo Sim4 e o Algoritmo 2 apresenta um pseudocódigo em alto nível.

Algoritmo 2 : Processamento paralelo de pares de seqüência.

```
Entrada: (1) p processadores p0, p1, ..., pp-1. (2) nome do Arquivo de Seqüências genômicas (R) e nome do Arquivo de cDNA (S)
Saída: O alinhamento spliced entre a seqüência genômica e o cDNA
ProcessaPares ()
início
        SETUP()
        MestreId ← 0

        taskId ← MyId()

        se taskId = MestreId então
                lista1 ← montaLista( R )
                m ← |lista1|
                divida o conjunto R em m arquivos temporários

                lista2 ← montaLista( S )
                n ← |lista2|
                divida o conjunto S em n arquivos temporários
        fim-se


        broadcast( lista1, pi )
        broadcast( lista2, pi )

        se taskId = MestreId então
                para i de 1 até |lista1| faça
                        para j de 1 até |lista2| faça
                                receive(id,request)
                                send (id, i)
                                send (id, j)
                        fim-para
                fim-para
                para i de 1 até ntask faça
                                receive(id,request)
                                send (id, -1)
                                send (id, -1)
                        fim-para
                fim-para
        senão
                faça
                        send(taskId, request)
                        receive(MestreId,i)
                        receive(MestreId,j)
                        se i ≠ -1 e j ≠ -1 então
                             Sim4(lista1[i], lista2 [j])
                             Armazena resultados
                        fim-se
                enquanto ((i ≠ -1) e (j ≠ -1))
        fim-se

        se taskId = MestreId então
                ApresentaResultado()
        fim-se
fim
```

A grande vantagem desta abordagem está no processamento de arquivos pequenos. Isto evita uma atividade de paginação de página excessiva da memória virtual, que poderia degradar o desempenho do sistema.

4.2 Outras abordagens analisadas

Para verificar se a abordagem apresentada na seção 4.1 é adequada, implementamos outras duas abordagens para analisar comparativamente o comportamento de desempenho. A seções seguintes descrevem estas abordagens.

4.2.1 Processamento paralelo de subconjuntos de R versus S

Na primeira abordagem alternativa, o mestre divide os conjuntos R e S em *m* e *n* arquivos temporários, respectivamente. Isto é feito por meio de um pré-processamento dos conjuntos. Cada arquivo armazena uma única seqüência a ser analisada.

Uma lista contendo os *m* nomes de arquivos do conjunto R é replicada a todos os escravos. Considere *p* como o número de processos escravos. Os elementos do conjunto S são distribuídos de maneira circular em *p* subconjuntos com *n/p* elementos. Esta estratégia assegura a distribuição uniforme de seqüências do conjunto S a serem processadas por cada escravo.

O processamento ocorre em paralelo na execução de processos que executam o Sim4. Um elemento do conjunto R é processado com um elemento do subconjunto de S. O resultado do processamento é armazenado no arquivo de saída.

No caso das seqüências possuírem tamanhos aproximados, esta abordagem pode ser apropriada. Mas, se existir algum elemento do conjunto S que tenha tamanho muito maior demais, o escravo ao qual este elemento for atribuído, executará o seu processamento durante um longo período de tempo, enquanto outros escravos que terminarem suas tarefas ficam ociosos. Ocorre, portanto, a degradação do sistema.

Na Figura 6, o mestre replica uma lista de arquivos do conjunto R. O conjunto S é dividido em dois subconjuntos. O primeiro subconjunto $\{s_1, s_3, s_5, ... s_{n-1}\}$ de S é enviado ao escravo 1 e o segundo subconjunto $\{s_2, s_4, s_6, ... s_n\}$ de S é enviado ao escravo 2. A combinação de *m* seqüências do conjunto R e *n/p* seqüências do conjunto S, resulta em processamento de O(*m*× *n/p*) pares de seqüências em cada escravo.

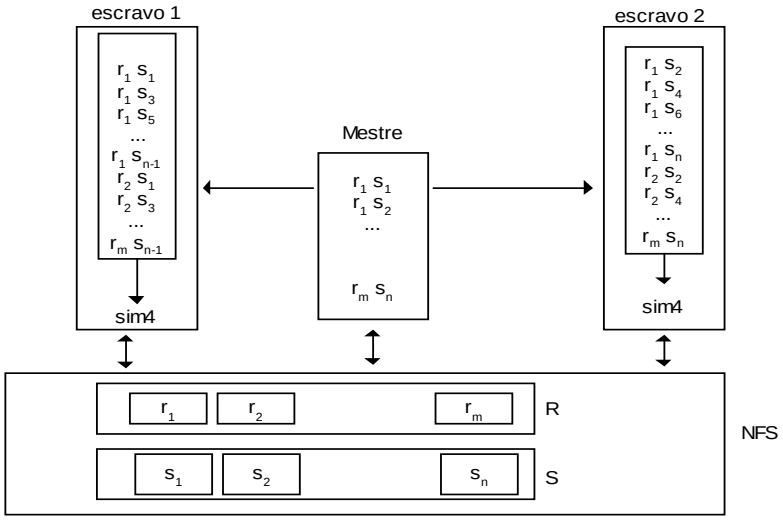

Figura 6: Processamento do subconjunto de R versus S.

4.2.2 Processamento paralelo de elementos de R versus conjunto S

Na segunda abordagem alternativa, o mestre divide o conjunto R em *m* arquivos temporários, por meio de um pré-processamento. Cada arquivo armazena uma única seqüência a ser analisada. O número de processos é *p*, para esta implementação *p* inclui o número de processos escravos e mais o processo mestre. O conjunto R é dividido em *p* subconjuntos com *m/p* elementos. Uma lista de nomes de arquivos com *m/p* elementos do subconjunto é distribuído para cada processo. Isto assegura a distribuição uniforme de tarefas entre os processos.

O conjunto S, que não é dividido em arquivos temporários, é replicado entre os processos. Cada elemento de cada subconjunto de R e o conjunto S servirão de parâmetros para execução do Sim4 nos processos escravos.

Em alguns casos, a divisão de *m* por *p* não é exata, ou seja, existem elementos de R que não são distribuídos aos processos. Neste caso, esta implementação trata este resíduo de maneira diferente. Ao terminarem a execução do Sim4, os processos solicitam outras novas tarefas. O mestre distribui cada elemento do resíduo a um processo solicitante, que passa o elemento do resíduo e o conjunto S como parâmetro ao Sim4. Note que este processamento é feito com um único elemento do resíduo e comparado ao conjunto S. Todos os resultados são gravados em um arquivo de saída. Após o processamento, o escravo novamente solicita outra tarefa. E assim por diante, até que não existam mais elementos do resíduo a serem processados. O mestre é o responsável por agrupar os resultados gerados pelos processos. A Figura 7 ilustra esta abordagem. Neste exemplo, o conjunto S foi replicado entre os nós e o conjunto R foi dividido em *m* arquivos e distribuídos de maneira uniforme entre os nós.

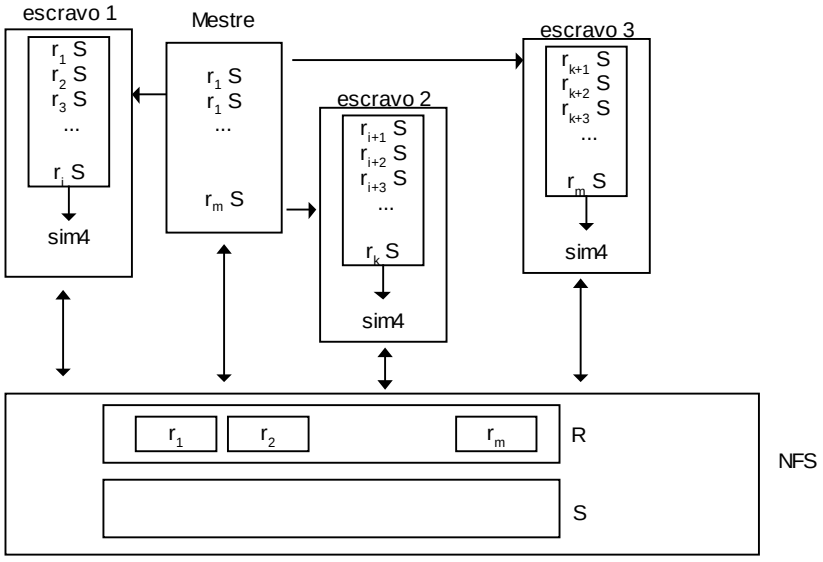

Figura 7: Processamento paralelo de elemento de R versus S.

5. Testes e Resultados

Nesta seção apresentamos os resultados obtidos com a implementação da versão paralela do Sim4. Os testes foram executados utilizando 1, 2, 4, 8 e 16 nós de processamento de um *cluster* disponível no Laboratório de Arquitetura e Software Básico da Escola Politécnica da Universidade de São Paulo (LASB-USP). Esta plataforma é composta pelo seguinte conjunto de máquinas interligadas por uma rede *Fast Ethernet* de 100 Mbits/s. Um aglomerado de 16 computadores dotados, cada um deles com um processador Intel Celeron, com *clock* de 433 MHz, de 128 MB de RAM e memória cache de 128 KB. Todos têm instalado linux RedHat 8.0 e LAM/MPI 6.5.9.

As seqüências genômicas e de cDNAs foram obtidas do banco de dados público *GenBank* no *site* (http://www.ncbi.nlm.nih.gov). Estas seqüências de DNA são provenientes do *Homo sapiens*. Para os testes, foram selecionados quarenta seqüências de cDNA e cem seqüências de DNA genômico.

Para cada teste, foram realizadas oito medições de tempo. O valor médio usado como referência para comparação representa a média aritmética dos tempos medidos.
A figura 8 apresenta os tempos de execução e *speed up* para os testes realizados, os gráficos mostram a comparação entre a solução proposta e outras duas abordagens implementadas.

Pode-se notar que com dois processadores na solução proposta, o processamento paralelo executa em tempo semelhante ao programa seqüencial. Isto ocorre porque o processo mestre, que não participa do processamento, envia tarefas a serem executadas para um único escravo. Contudo, com o aumento do número de processadores, o tempo de execução diminui proporcionalmente. Com mais processadores, existem mais escravos para realizar as tarefas executadas por demanda. A distribuição dinâmica de tarefas proporciona um balanceamento de carga de processamento.
A primeira abordagem alternativa, para os casos de testes realizados, não apresenta bons resultados. Alguns processos que terminam suas tarefas ficam ociosos, enquanto outros continuam processando, o que degrada o sistema. Isto ocorreu porque, particularmente neste teste, um elemento do conjunto S era muito maior que os demais. O processamento deste elemento S em um determinado escravo tem um longo tempo de execução, enquanto os outros escravos que terminam suas tarefas ficam ociosos. Ao contrário da abordagem proposta, nesta primeira abordagem alternativa não ocorre balanceamento de carga.

A segunda abordagem alternativa apresenta bons resultados para todos os testes. Apresenta melhores desempenhos para 2 e 4 processadores em relação a abordagem proposta. Contudo, com quantidades maiores de processadores a abordagem proposta obtém maiores ganhos de desempenho.

Outro aspecto importante a ser considerado é o tamanho dos arquivos de entrada. Na terceira abordagem um elemento do conjunto R é comparado com o arquivo que contém todo o conjunto S. Se esse conjunto S for muito grande ocorrem muitas trocas de páginas em memória virtual, o qual degrada o desempenho do sistema. Isto não acontece na primeira abordagem, já que o processamento é feito sobre pequenos arquivos.

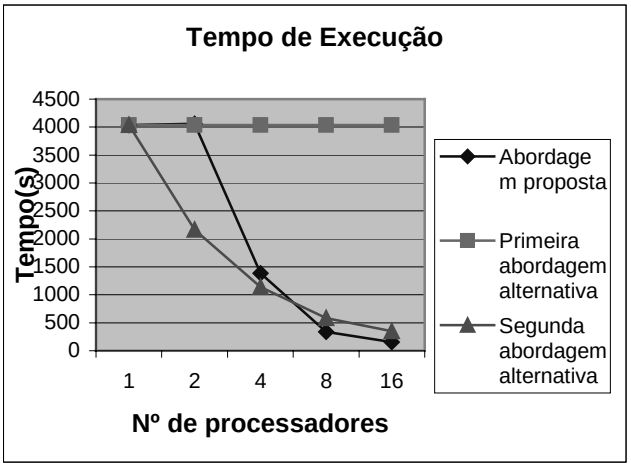

Figura 8 (a): Tempo de execução

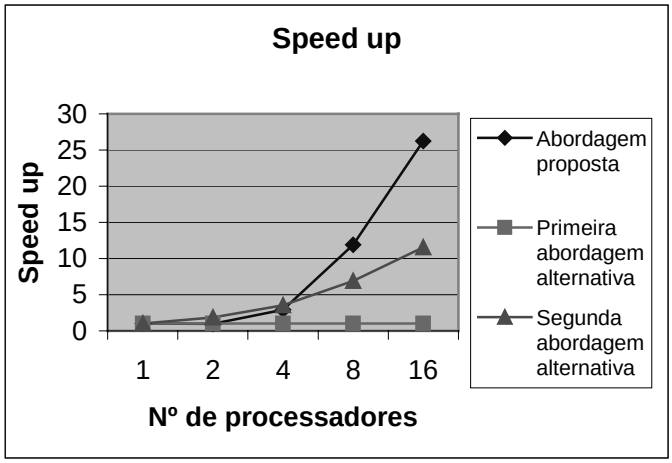

Figura 8 (b): Curva *Speed up*

6. Conclusão

Devido a grande quantidade de dados biológicos disponíveis em banco de dados públicos, algoritmos que utilizam abordagens extrínsecas para determinar regiões componentes do DNA exigem grande quantidade de processamento. Assim, algoritmos de predição de genes, que realizam análises e identificam a estrutura de um gene, estão associados a longos tempos de execução. Isto requer a criação de ferramentas mais rápidas para o tratamento destes dados.

Neste artigo apresentamos uma solução paralela para o tratamento do problema de reconhecimento de genes utilizando o algoritmo Sim4, o qual produz o alinhamento de um cDNA com um DNA genômico levando em consideração os limites entre éxons e íntrons. A execução simultânea de processos que executam o Sim4 proporcionam ganho de desempenho em alinhamentos de grande quantidade de dados biológicos para resolver o problema do alinhamento

spliced. O principal fator deste aumento de desempenho se deve a independência dos dados nas execuções locais e a estratégia de distribuição de dados entre os processadores. Outro fator se deve a utilização do sistema de arquivo de rede que dispensa a troca de mensagens grandes e tráfego pela rede.

Como trabalhos futuros pretende-se analisar outras estratégias de distribuição de dados, como por exemplo, distribuição de dados em ambientes sem disco compartilhado. Com estas análises poder-se-ia entender como a transferência de dados biológicos de tamanho desconhecido poderia influenciar no desempenho.

Referências

[1] Altschul, S. F., Gish, W., Myers, E. W. e Lipman, D. J. *Basic local alignment search tool. Journal of Molecular Biology*, 215:403-410, 1990.

[2] Borodovsky, M e McIninch, J. *GENMARK: parallel gene recongnition for both DNA strands*. Comp. Chem. 17, 123-133.

[3] Chao, K. M., Zhang, J., Ostell, J. and Miller, W. *A tool for aligning very similar sequence*. In Comput. Appl. Biosci., volume 13, pages 75-80. 1997.

[4] Florea, L., Hartzell, G. Zhang Z., Rubin, G. M., e Miller, W. *A computer program for aligning a cDNA sequence with a genomic sequence, Genome Research,* 8(9):967-974,1998.

[5] Guigó, R. acesso feito em 30 de maio de 2004 http://www.pdg.cnb.uam.es/cursos/BioInfo2001/pages/GenId/GeneIdentification/TheProblem.html.

[6] Guigó, R., Knudsen, S., Drake, N. e Smith, T. *Prediction of gene structure*. J. Mol. Biol. 226, pages 161-157, 1992.

[7] Gusfield, D. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, Cambridge,1997. Computer science and computacional biology

[8] Gelfand, M. S., Mironov A. A. e Pevzner P. A. *Gene recognition via spliced sequence alignment. Proc. Natl. Acad. Sci USA*, 93:9061-9066, 1996.

[9] Gropp, W., Lusk, E. and Skjellum, A., *Using MPI: portable parallel programming with the Message Passing Interface*. MIT Press, Cambridge, Massachusetts, USA, 1994.

[10] Kent W. J., *BLAT - The BLAST-like alignment tool. Genome Res.*, 12:656-664,2002.

[11] Kulp, D., Haussler, D., Reese, M. G. e Eeckman, F. H. *A generalized Hidden Markov Model for the recognition of human genes in DNA. In Proceeding of the Fourth International Conference on Intelligent System for Molecular Biology. AAAI Press, Menlo Park, CA.*, 1996.

[12] Mathé C., Sagot M. Schiex T., Rouzé P. "*Current method of gene prediction, theis strenght and weakness*", *Nucleic Acids Research*, 2002, Vol. 30 No.19, pages 4103-4117, 2002.

[13] Meidanis, J., Setúbal, J. C., *Introduction to computacional molecular biology*, PWS Publishing Company University of Campinas, Brazil, 1997.

[14] Mott, R., *EST_GENOME: A program to align spliced DNA sequences to unspliced genomic DNA. Comput. Appl. Biosci.* 13:477-478, 1997.

[15] Needleman, S. B., Wunsch, C. D., *A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology,* 48:443-453, 1970.

[16] Pearson W. R., Lipman D. J. *Identification of common molecular subsequences. Journal of Molecular Biology*, 147:195-197.

[17] Smith, T. F., Waterman, M. S., *Identification of common molecular subsequences. Journal of Molecular Biology*, 147:195-197, 1981.

[18] Solovyev, V. V., Salamov, A. A. & Lawrence, C. B. *Predicting internal exons by oligonucleotide composition and discriminant analysis of spliceable open reading frames.* Nucl. Acid. Res. 22, pages 5156-5163., 1994.

[19] Wheelan, S. J., Church, D. M., e Ostell, J. M. *Spidey: A tool for mRNA-to-genomic alignment. Genome Research.* 11: 1952-1957, 2001.