ESTRATÉGIAS DE PARALELISMO NA BUSCA POR SIMILARIDADES EM SEQÜENCIAS DE DNA E PROTEÍNAS JUNTO AO MPIBLAST

Yura Carvalho Ferreira

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Profª. Marta Lima de Queirós Mattoso, D.Sc.

Prof. Geraldo Bonorino Xexéo, D.Sc.

Prof. Alberto Matín Rivera Dávila, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

AGOSTO DE 2005

FERREIRA, YURA CARVALHO

Estratégias de Paralelismo na Busca por Similaridades em Seqüências de DNA e Proteínas Junto ao mpiBLAST. [Rio de Janeiro] 2005

XII, 104 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2005)

Dissertação - Universidade Federal do Rio de Janeiro, COPPE

1. BioInformática 2. Paralelismo 3. Banco de Dados 4. BLAST 5. Alinhamento de Seqüências

I. COPPE/UFRJ II. Título (série)

Aos meus pais, Aloysio e Herondina

Aos meus irmãos, Fernanda, Rodrigo e Baruch

À minha namorada, Sharen

AGRADECIMENTOS

À Professora Marta Mattoso, por ter me dado a oportunidade de fazer pesquisa científica nesta nova área interdiciplinar, a BioInformática, que me desperta tanto interesse. Pela sua orientação, sua dedicação e por me disponibilizar parte de seu precioso tempo nestas atividades.

Ao Professor Alberto Dávila, por seu entusiasmo com o projeto, pelos preciosos conhecimentos de biologia passados, pela confiança, disponibilidade e dedicação.

À Professora Myrian, pelas inúmeras vezes que me ajudou durante o processo de execução dos experimentos, privando-se, inclusive, de seu tempo de descanso.

Aos Comandantes da Marinha, Frederico e Gomes Lima, por terem sempre me apoiado e ajudado em todos os sentidos: pessoal, acadêmico, e profissional. Pela confiança depositada em minha pessoa, pela paciência com minhas atitudes, pelas "licenças de trabalho" fundamentais para que eu pudesse fazer e concluir este mestrado, pelo valor dado ao meu trabalho, em fim, por tudo isso, muito obrigado.

Ao Professor Blaschek, por sempre me ter incentivado, por diversas vezes me acalmado com seus conselhos e experiência de vida, sempre sendo positivo e paciente. Por me ter dado as oportunidades nos projetos, viabilizando meus estudos, minha vida pessoal e profissional. Como você sempre diz "Sustentai o fogo que a vitória é nossa!".

Aos meus colegas da COPPETEC e da equipe do CECAFA, principalmente Bruno Kinder, Cristiane, Leonardo, Lúcio, Mario Anibal, Renato e Ricardo, pelo espírito de equipe, amizade, pelas brincadeiras e pelo apoio e vontade de ajudar em tudo que fosse necessário.

A CAPES pelo suporte financeiro a este trabalho.

Aos colegas de mestrado e doutorado, em especial ao meu amigo André Ormastroni, por sempre me ter ajudado, cooperado e sempre se ter colocado disponível.

Aos meus pais e irmãos, que sempre estiveram comigo e me permitiram ser quem sou hoje.

Aos meus amigos André e Mário, por todos esses anos de amizade, brigas, maluquices e diversão.

Enfim, gostaria de finalizar agradecendo a todos que fizeram e fazem parte da minha vida e que de uma forma ou de outra contribuíram para eu estar aqui.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ESTRATÉGIAS DE PARALELISMO NA BUSCA POR SIMILARIDADES EM SEQÜÊNCIAS DE DNA E PROTEÍNAS JUNTO AO MPIBLAST

Yura Carvalho Ferreira

Agosto/2005

Orientadora: Marta Lima de Queirós Mattoso

Programa: Engenharia de Sistemas e Computação

Esta dissertação analisa técnicas de bioinformática aplicadas na busca por similaridade de seqüências de DNA e de proteínas. Em especial, é estudada a principal ferramenta de busca baseada em alinhamento local de seqüências, denominada BLAST (Basic Local Alignment Search Tool). Embora o BLAST seja uma das ferramentas mais eficientes dentre as existentes, o número elevado de seqüências a serem comparadas faz com que o tempo de processamento seja muito alto e chegue a durar semanas em alguns casos. Uma das técnicas de computação adequada nesses casos é o processamento paralelo. Porém, essas ferramentas são em geral "caixas-pretas" e para efeitos de comparação é importante que todos utilizem o mesmo algoritmo. Assim, ao invés de realizar uma paralelização no código fonte do BLAST, uma alternativa interessante é a obtenção do paralelismo através da distribuição de dados. Esta dissertação apresenta uma avaliação de estratégias de paralelismo junto ao BLAST e em especial realiza uma análise do mpiBLAST, que é um projeto de software livre que provê componentes para a execução paralela do BLAST. Como contribuição, a dissertação propõe alternativas de distribuição de dados para execução paralela junto ao mpiBLAST de modo a auxiliar o usuário na configuração do paralelismo mais adequado à sua busca.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

PARALLEL STRATEGIES WITH MPIBLAST FOR SEARCHING SIMILARITIES IN DNA AND PROTEIN SEQUENCES

Yura Carvalho Ferreira

August/2005

Advisors: Marta Lima de Queirós Mattoso

Department: Systems and Computer Engineering

This dissertation analyzes bioinformatics' techniques applied in the search for protein and DNA sequences similarity. In special, the main tool for searching based in local alignment of sequences, called BLAST (Basic Local Alignment Search Tool). Although BLAST's one of the most efficient tools amongst existing ones, the high number of sequences to be compared lead to a high processing time, and in some cases it may last weeks. In these cases, one of the appropriated computation techniques is the parallel processing. In general, these tools are "black-box" and for comparison purposes it is important that everyone uses the same algorithm. Thus, instead of parallelizing the BLAST's source code, an interesting alternative is to achieve parallelism through the data distribution. This dissertation presents an evaluation of parallel strategies applied to BLAST. In special the open source project mpiBLAST, which offers components to the parallel execution of BLAST is analyzed. As major contributions this dissertation proposes data distribution alternatives to the parallel execution of BLAST in order to support the configuration of the appropriated parallelism to the user's search.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

1 Introdução

A descoberta da estrutura do DNA em 1953 mudou significativamente a maneira pela qual a biologia é estudada. Em um esforço conjunto, biólogos do mundo inteiro estão tentando decifrar o DNA existente em cada forma de vida presente no planeta, produzindo assim uma extraordinária quantidade de dados que necessitam ser analisados. Desta forma, a biologia está se tornando cada vez mais uma ciência que necessita processar um grande volume de dados de forma a obter informações relevantes a respeito de seus questionamentos. Esta é uma das principais razões para o surgimento de uma nova área interdisciplinar, denominada BioInformática. A BioInformática combina os conhecimentos da Biologia e das Ciências da Computação na tentativa de construir ferramentas que dêem o suporte computacional necessário para que os biólogos possam obter ou inferir conhecimentos biológicos significantes [1].

Uma das principais formas de se aprender algo a respeito de novas seqüências obtidas é compará-las com outras seqüências previamente estudadas, cuja função já seja bem conhecida. A comparação entre seqüências biomoleculares é um dos principais problemas em bioinformática, e em geral este é resolvido através da técnica de alinhamento de seqüências. Dentre as principais ferramentas utilizadas para obtenção de alinhamentos, encontramos o BLAST [2, 3]. O BLAST ("Basic Local Alignment Search Tool") é em geral a primeira ferramenta utilizada por um biólogo quando este obtém uma nova seqüência de proteínas ou nucleotídeos. Basicamente o BLAST compara a nova seqüência obtida, com todas as seqüências existentes em uma ou mais bases de dados, retornando as que forem mais similares a esta.

O algoritmo do BLAST foi inicialmente desenvolvido pelo NCBI ("National Center for Biotechnology Information") e desde então diversas outras implementações dele foram feitas. Desde a sua primeira

versão, lançada em 1990, um dos principais objetivos a ser alcançado em seu desenvolvimento foi o de conseguir executar as consultas o mais rápido possível, sem sacrificar a sensibilidade do algoritmo. Apesar de sua grande eficiência, o crescimento vertiginoso das bases de dados tem elevado em muito o tempo de resposta das requisições feitas a ele. Como alternativa para obtenção de melhor desempenho, a utilização de ambientes paralelos para execução do BLAST tem sido bastante aplicada, principalmente em ambientes de memória distribuída, como "clusters" de microcomputadores. Neste contexto uma ferramenta que tem obtido bastante destaque é o mpiBLAST [4].

O mpiBLAST é um projeto de software livre, com código fonte aberto, que implementa a execução em paralelo do NCBI-BLAST [5]. A estratégia de paralelismo do mpiBLAST consiste em fragmentar a base de dados, distribuí-la entre os nós de um "cluster", e executar a mesma consulta BLAST, simultaneamente em cada nó; cada um realizando a busca sobre uma porção única da base de dados original. Tal estratégia por fragmentar a base de dados, pode permitir a alocação desta inteiramente em memória principal, acelerando em muito a execução da consulta.

Esta dissertação é motivada por dois projetos de pesquisa em bioinformática, o BioWebDB [6] e o Bats [7]. Ambos projetos realizam consultas ao BLAST que demandam um tempo muito grande de processsamento, podendo durar cerca de um mês. Além disso, os projetos se organizam a partir de consórcios que compartilham recursos de hardware paralelo e fazem uso intensivo de software de código aberto. O objetivo desta dissertação é apresentar uma solução para o problema de desempenho em consultas ao BLAST. Apesar de haver diversas propostas de paralelismo para o BLAST, observamos que as soluções ou dependem de hardware específico ou são soluções de software proprietário. Assim, esta dissertação contribui ao fazer uma análise do mpiBLAST e baseado nas estratégias de paralelismo já existentes, propor algumas otimizações. Desta forma, nos baseamos em hardware de prateleira e software de

código aberto, além de disponibilizar publicamente nossa solução. Inicialmente foi realizado um estudo minucioso do comportamento do BLAST bem como o relacionamento deste com determinados aspectos biológicos. Dentre os principais aspectos a serem abordados, destacamos, a análise de estratégias de paralelismo que interferem na distribuição da carga de trabalho e a detecção de etapas que caracterizam gargalos na execução do sistema. Para poder avaliar estas questões, além do estudo das técnicas ligadas ao paralelismo, inúmeros experimentos foram realizados através de execuções do mpiBLAST e do NCBI-BLAST [5]. A escolha das seqüências da consulta e da base de dados, a serem utilizadas nos experimentos, foi feita de modo que se evitasse a geração de dados aleatórios que pudessem não refletir uma situação real de uso da ferramenta. Assim, as seqüências de consulta foram obtidas de dados reais de pesquisa da Fundação Oswaldo Cruz, no Rio de Janeiro. A base de dados utilizada foi a do *nt*[*], por ser uma das maiores bases de dados públicas disponíveis. Os tipos de busca BLAST utilizados foram: o tBLASTx, por exigir maior quantidade de recursos computacionais e tempo de processamento e o BLASTn, por ser o outro tipo de busca BLAST adequado as seqüências escolhidas para consulta e para base de dados. Os resultados obtidos revelaram aspectos importantes para estratégias de fragmentação da base de dados bem como ganho de desempenho de até 30% em relação a estratégia atual do mpiBLAST.

Esta dissertação está organizada da seguinte forma. No Capítulo 2 são apresentados os conceitos principais relativos à Biologia Molecular, ao BLAST e à execução do BLAST em ambientes paralelos. No Capítulo 3 é feita uma descrição detalhada do funcionamento do mpiBLAST. No Capítulo 4 o mpiBLAST é analisado e são apresentadas as principais propostas desta dissertação com seus resultados experimentais analisados. Finalmente no Capítulo 5 são apresentadas as principais conclusões deste trabalho, bem como suas contribuições para esta área de pesquisa.

[*] ftp.ncbi.nih.gov/blast/db/FASTA/nt.gz

2 Processamento Paralelo de Análises em Biologia Molecular

O objetivo deste capítulo é apresentar os principais conceitos e assuntos relacionados ao tema desta dissertação.

Na seção 2.1 é apresentada uma breve introdução a biologia molecular. A biologia molecular é o ramo da biologia que procura compreender a estrutura e a função das proteínas e ácidos nucléicos. Dentre os diversos métodos e técnicas de compreensão dessa estrutura e função está o ramo de pesquisa em bioinformática, onde métodos computacionais são utilizados de forma complementar às análises laboratoriais. Apesar de não haver um consenso em relação ao termo bioinformática, este pode ser definido como a utilização em conjunto de métodos matemáticos, estatísticos e computacionais, com o objetivo de desenvolver ferramentas que sejam capazes de analisar seqüências biomoleculares e informações relacionadas e assim, responder determinadas questões biológicas [8].

Na seção 2.2 é apresentado e descrito um dos principais instrumentos computacionais para análise e comparação de seqüências, o BLAST. O BLAST de "Basic Local Alignment Search Tool" [2, 3] é tipicamente a primeira ferramenta de bioinformática utilizada por um biólogo, quando este examina uma nova seqüência de nucleotídeos ou proteínas obtida. Basicamente, o que o BLAST faz é comparar uma nova seqüência desconhecida, com todas as seqüências pré-analisadas e depositadas em uma base de dados conhecida, retornando as que forem mais similares a esta.

Finalmente, na seção 2.3 são apresentados alguns conceitos básicos de processamento paralelo e paralelismo em banco de dados, e as principais aplicações de estratégias dessas áreas no BLAST.

2.1 Breve Introdução à Biologia Molecular

Na natureza encontramos seres vivos e seres inanimados. Em geral, os seres vivos podem se mover, se reproduzir, crescer, comer, etc. – eles possuem uma participação ativa em seu meio, o que não ocorre com os seres inanimados. Apesar disso, pesquisas já revelaram que ambos são compostos por átomos, estando assim sujeitos às mesmas regras da química e da física. Então o que realmente os diferencia? Por um longo tempo na história humana, pessoas imaginavam que haveria alguma substância extra que concederia aos seres vivos sua característica ativa. Porém tal substância nunca foi encontrada. De fato, o que atualmente se entende a respeito disto é que os seres vivos possuem esta característica ativa devido às complexas reações químicas que ocorrem dentro deles. Estas reações nunca cessam, e em geral, os produtos de uma reação são constantemente consumidos por outras reações, mantendo assim o sistema ativo. Um organismo vivo também está constantemente trocando substâncias e energia com seu meio. De maneira oposta, um ser que não está fazendo isto poderia ser considerado morto – algumas exceções seriam as sementes e os vírus, que podem permanecer inativos por um longo período de tempo sem estarem mortos [9, 10].

A ciência moderna afirma que a vida na terra surgiu há cerca de 3,5 bilhões de anos atrás. A primeira forma de vida era bastante simples, porém após bilhões de anos, um processo contínuo, chamado *evolução*, fez com que esta forma simples se desenvolvesse e se diversificasse, de maneira que hoje encontramos tanto organismos bastante complexos como outros bastante simples.

Ambos os organismos, complexos e simples, possuem uma química ou bioquímica molecular semelhante. Os principais atores na química da vida são moléculas chamadas *proteínas* e *ácidos nucleicos*. Explicando de uma maneira simplificada, as proteínas são responsáveis pelo que somos e fazemos no sentido físico, já os ácidos nucleicos são

responsáveis por passar esta *receita* ou *características hereditárias* adiante para as outras gerações. Originalmente, suspeitava-se que o material hereditário era a proteína, porém em meados do século XX se tornou aparente que eram os ácidos nucleicos que continham tal material e que a proteína era sintetizada a partir dele, dentro da célula [11].

A biologia molecular [12] é o ramo da biologia que procura compreender a estrutura e a função das proteínas e ácidos nucleicos. A genética por sua vez é o ramo da biologia que estuda e procura explicar os fenômenos relacionados à hereditariedade. A pesquisa em biologia molecular tem como principal objetivo determinar os genomas completos de várias espécies, através do seqüênciamento de cadeias de DNA e da identificação da localização e das funções dos genes.

2.1.1 As Proteínas

Muitas das substâncias presente em nossos corpos são proteínas, dos quais existem diferentes tipos. Por exemplo, as *proteínas estruturais* - como a queratina e o colágeno - agem como blocos para construção dos tecidos, já um outro tipo de proteína denominada *enzima*, age como um catalisador de reações químicas. Um catalisador é uma substância que acelera as reações químicas. Muitas destas reações se não sofressem a ação de uma enzima, levariam um longo tempo para ocorrer ou até mesmo nunca ocorreriam e assim não teriam utilidade para a vida. As enzimas são bastante específicas, atuando em geral em apenas um tipo de reação. Considerando a grande quantidade de reações químicas necessárias para sustentar a vida, é necessário também uma grande variedade de tipos de enzimas. Como exemplo de outras funções das proteínas teríamos o transporte de oxigênio e os anticorpos de defesa. Mas o que seriam exatamente as proteínas? Do que elas seriam feitas? E como elas desempenham suas funções?

A proteína é formada por uma cadeia de moléculas simples chamadas *aminoácidos* (Figura 1).

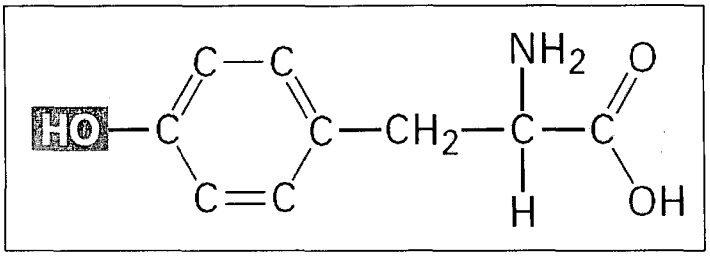

Figura 1 - Aminoácido Tirosina [13]

Na natureza encontramos 20 aminoácidos diferentes (Tabela 1); excepcionalmente alguns outros aminoácidos fora deste padrão podem aparecer.

Tabela 1 - Descrição dos 20 aminoácidos

Aminoácido	Abreviação	Símbolo	Aminoácido	Abreviação	Símbolo
Alanine	Ala	**A**	Methionine	Met	**M**
Cysteine	Cys	**C**	Asparagine	Asn	**N**
Aspartate	Asp	**D**	Proline	Pro	**P**
Glutamate	Glu	**E**	Glutamine	Gln	**Q**
Phenylalanine	Phe	**F**	Arginine	Arg	**R**
Glycine	Gly	**G**	Serine	Ser	**S**
Histidine	His	**H**	Threonine	Thr	**T**
Isoleucine	Ile	**I**	Valine	Val	**V**
Lysine	Lys	**K**	Tryptophan	Trp	**W**
Leucine	Leu	**L**	Tyrosine	Tyr	**Y**

Em uma proteína, os aminoácidos se ligam por *pontes de peptídeos* e por razão disto são denominados *cadeias polipeptídicas*. Ao se fazer a ligação entre dois aminoácidos, uma molécula de água é liberada (Figura 2). Assim, o que realmente existe na proteína é uma cadeia de *resíduos* dos aminoácidos originais. Por isso, é bastante comum falar que a proteína possui 100 resíduos, ao invés de 100 aminoácidos. Tipicamente uma proteína possui cerca de 300 resíduos, mas existem proteínas com menos de 100 e outras com quase 5000 resíduos. Como em uma das extremidades da cadeia peptídica há um *grupo amino* (NH_2) e na outra

um *grupo carboxyl* (COOH), pode ser dada uma direção à cadeia. Por convenção a cadeia começa no grupo amino "N-terminal" e termina no grupo carboxyl "C-terminal" (Figura 2). O conhecimento desta convenção é importante para se poder entender determinados resultados de algoritmos que realizam alinhamento de seqüências e buscas por similaridades (como por exemplo, o BLAST). A orientação N→C é representada com um '+' e a C → N com um '-'. Neste caso específico, o BLAST assume que a orientação apresentada nos arquivos de consulta e na base de dados é o positivo.

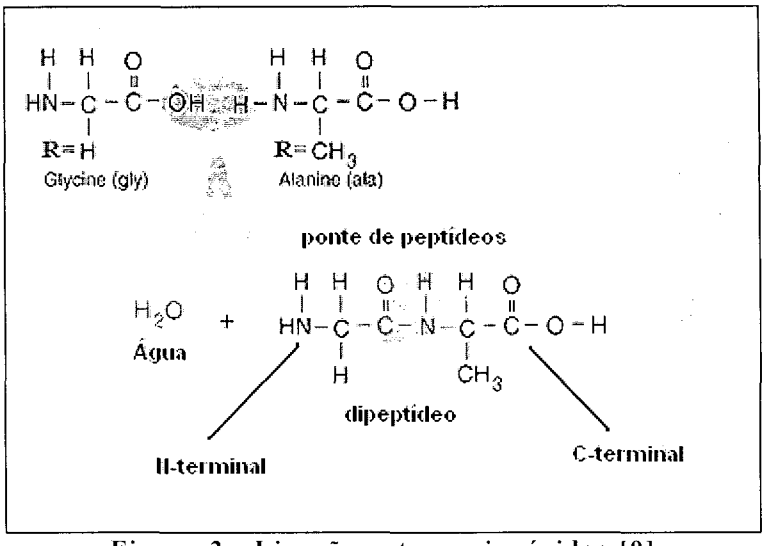

Figura 2 - Ligação entre aminoácidos [9]

A proteína não é apenas uma seqüência linear de resíduos. Na verdade, esta é a sua estrutura *primária*, existem ainda mais três estruturas ligadas à proteína, a *secundária*, a *terciária* e *quaternária*. Estas últimas, basicamente determinam à estrutura tridimensional da proteína, que é fundamental para determinar sua função. Apesar de serem bastante importantes para a pesquisa em biologia molecular, para os propósitos do BLAST apenas a estrutura primária é relevante.

As proteínas são produzidas em uma estrutura celular chamada *ribossomo*. Nos ribossomos, os componentes da proteína – os aminoácidos – são montados um a um, graças às informações contidas em uma molécula bastante importante chamada *ácido ribonucléico mensageiro*, ARNm ou RNAm (do inglês).

2.1.2 Os Ácidos Nucleicos

Os organismos vivos possuem dois tipos de ácidos nucleicos: o *acido ribonucléico*, ARN ou RNA (do inglês), e o *ácido desoxirribonucléico*, ADN ou DNA (do ingês), ambos detalhados nas sub-seções a seguir.

2.1.2.1 O DNA

Como a proteína, o DNA é uma cadeia de moléculas mas composto por nucleotídeos. Na verdade o DNA é formado por uma cadeia dupla, porém iniciaremos sua descrição pela cadeia simples, denominada *fita* ou "strand". Cada cadeia é composta pela repetição de uma mesma unidade básica. Esta unidade é formada por uma molécula de açúcar chamada *2'-desoxirribose* ligada a um resíduo de fosfato. A molécula de açúcar possui cinco átomos de carbono, e estes são rotulados de *1'* a *5'* (Figura 3). A ponte formada entre as unidades – que compõem a cadeia – são estabelecidas entre o carbono 3' de uma unidade, o resíduo de fosfato, e o carbono 5' da unidade seguinte. Por esta razão, de maneira análoga às moléculas de proteína, as moléculas de DNA também possuem uma orientação, que por convenção começa no carbono 5' e termina no carbono 3'. Quando observamos uma única fita de DNA em artigos técnicos, livros, etc., esta é escrita na direção 5' → 3', a menos que o contrário seja dito. Novamente, como ocorre com as proteínas, no BLAST esta convenção é apresentada com um sinal de mais '+' para indicar o sentido 5' → 3' e com um sinal de menos '-' o sentido oposto. No caso em específico, o BLAST supõe que a orientação apresentada nos arquivos de consulta e nas bases de dados é o positivo.

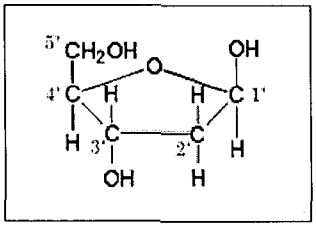

Figura 3 - 2'-desoxirribose

Ligadas ao carbono 1' de cada unidade presente na cadeia estão outras moléculas denominadas *bases*. Existem quatro tipos de bases: *adenina* (*A*), *guanina* (*G*), *citosina* (*C*) e *timina* (*T*). As bases A e G pertencem a um grupo de substâncias chamadas *purinas*, já as bases C e T ao grupo das *pirimidinas* (Figura 4).

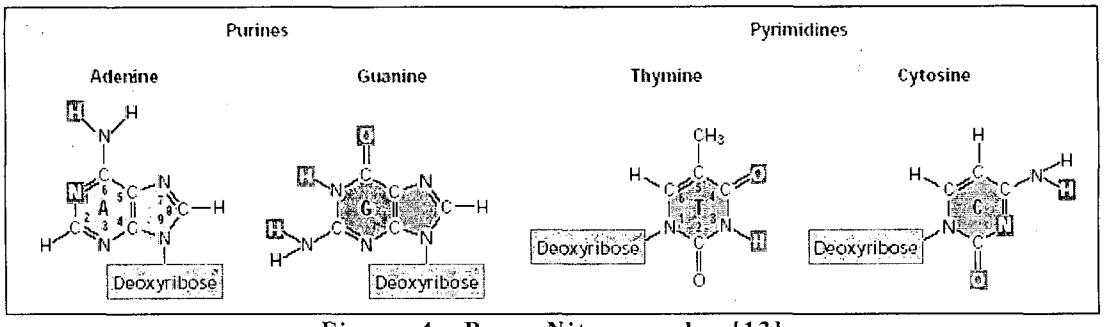

Figura 4 - Bases Nitrogenadas [13]

Quando as unidades básicas da molécula de DNA são vistas como consistindo de açúcar, fosfato, e sua base (Figura 5), estas são denominadas *nucleotídeos*. Portanto, embora bases e nucleotídeos não sejam exatamente a mesma substância, pode-se dizer que uma molécula de DNA possui 200 bases ou 200 nucleotídeos. Em geral, as moléculas de DNA são bem grandes, muito maiores do que as proteínas. Em células humanas, as moléculas de DNA possuem centenas de milhões de nucleotídeos. Na Figura 6, estão esquematizadas as ligações entre os nucleotídeos, em (a) é apresentado uma versão detalhada mostrando os átomos em cada ligação química e em (b) uma versão mais resumida, mostrando apenas os principais elementos.

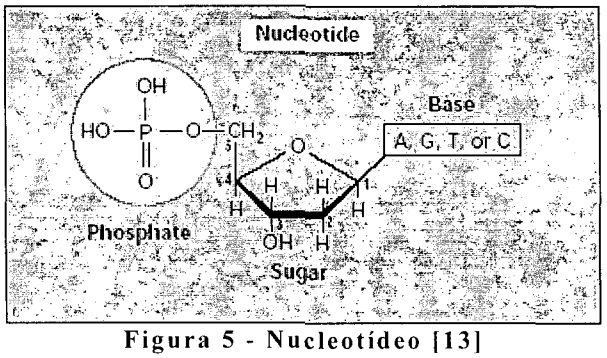

Figura 5 - Nucleotídeo [13]

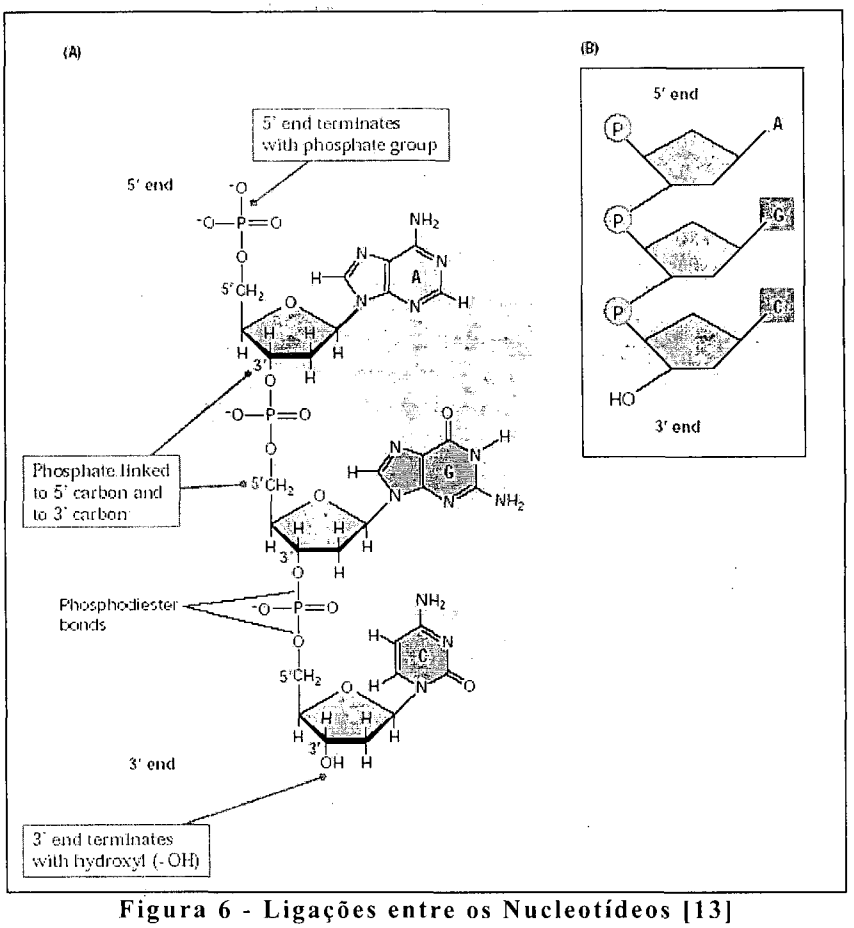

Figura 6 - Ligações entre os Nucleotídeos [13]

Como já mencionado anteriormente, as moléculas de DNA são compostas por duas cadeias ou "strands", que se mantêm juntas na famosa estrutura de *dupla-hélice* descoberta por James Watson e Francis Crick em 1953 (Figura 7). As cadeias se mantêm ligadas através das bases nitrogenadas. A base 'A' faz par com a base 'T' e a base 'C' faz par com a base 'G'. As bases A e T, são ditas complementares uma a outra ou um par de *bases complementares*. De maneira análoga, C é complementar a G. Estas bases são conhecidas como "Watson-Crick base pairs" ou apenas *pares de base*. Os pares de base são freqüentemente utilizados como unidade de comprimento das moléculas de DNA. Utilizando a forma abreviada *pb* pode-se dizer, por exemplo, que um pedaço da molécula de DNA possui 100 pb ou 100.000 pb ou 100 kpb de comprimento.

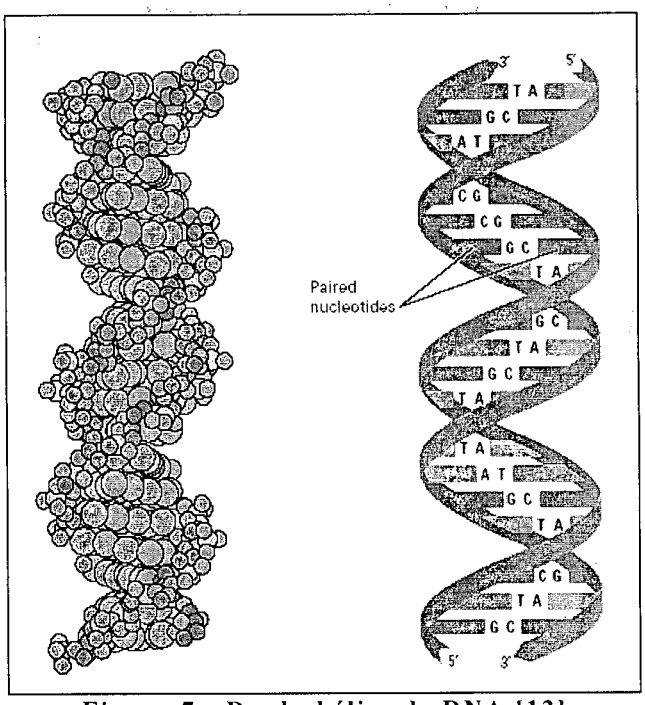

Figura 7 - Dupla-hélice de DNA [13]

Para muitos propósitos, inclusive para o BLAST, o DNA é considerado uma cadeia de caracteres ou letras, onde cada letra representa uma das bases nitrogenadas. Embora ambas as cadeias estejam ligadas, é preservada a orientação de cada uma. A orientação de uma cadeia é oposta à orientação da outra, assim o extremo 3' de uma corresponde ao extremo 5' da outra (Figura 8). Esta propriedade é expressa dizendo-se que as duas cadeias são *antiparalelas*. A conseqüência fundamental desta estrutura é que se torna possível inferir – por complementação – a seqüência de uma cadeia dado a cadeia oposta.

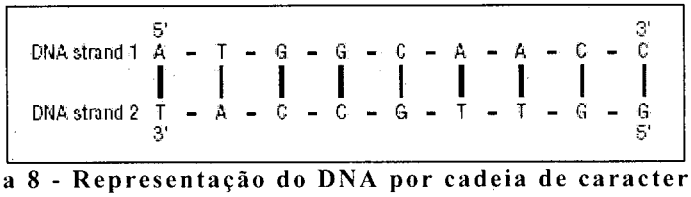

Figura 8 - Representação do DNA por cadeia de caracteres [1]

É precisamente este mecanismo que permite a replicação do DNA presente na célula. DNA este que contêm as informações genéticas necessárias para especificar a construção de todo um organismo [12]. Assim, um ser que inicia sua vida com apenas uma célula, pode crescer,

chegando a possuir bilhões de células, cada uma delas contendo uma cópia da molécula de DNA presente na célula original (Figura 9).

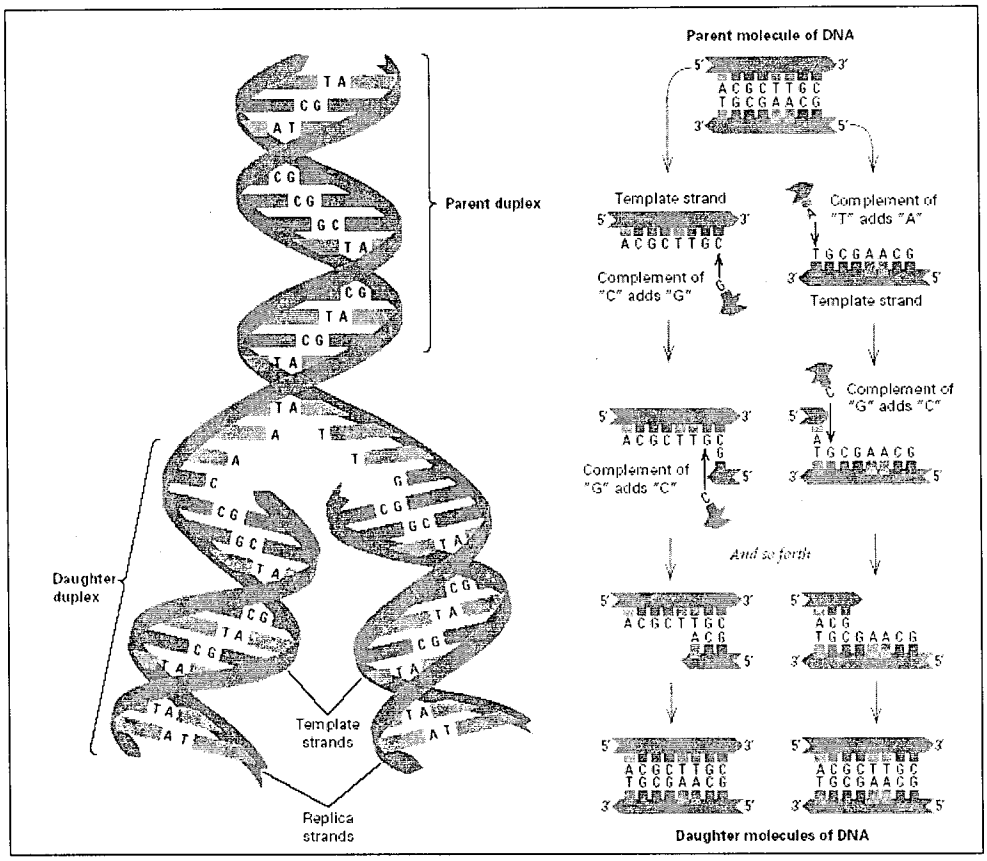

Figura 9 - Duplicação do DNA [13]

Em organismos cuja célula não possui núcleo, o DNA se encontra disperso no citoplasma (procariotos). Em organismos superiores, o DNA se encontra dentro de organelas e o núcleo (eucariotos).

2.1.2.2 O RNA

Moléculas de RNA são muito parecidas com as moléculas de DNA. As principais diferenças são as seguintes:

- No RNA o açúcar é a ribose, ao invés da 2'-desoxirribose.
- No RNA não se encontra a base timina; ao invés disso encontra-se a base *uracila* (*U*), e esta faz par com a adenina.

- O RNA não forma uma dupla hélice. Às vezes o RNA se liga ao DNA formando uma hélice híbrida.

Uma outra diferença entre o DNA e o RNA é que enquanto o DNA desempenha essencialmente uma função, armazenar informação codificada, o RNA, com suas diferentes formas, executa diversas funções dentro da célula.

2.1.3 Expressão Gênica e o Dogma Central da Biologia Molecular

Como já se sabe, a informação necessária para construir as proteínas está contida nas moléculas de DNA. Por esta razão, o DNA é mencionado como "the blueprint of life" ou *a planta de construção da vida* [13]. Em cada célula de um organismo existem algumas, e bem longas, moléculas de DNA. Tais moléculas são chamadas *cromossomos*. Nem toda informação contida na cadeia de DNA é destinada à construção de proteínas, na verdade, são alguns trechos contínuos do DNA que guardam tal informação. Em geral, a informação necessária para codificar cada tipo de proteína aparece em um único trecho contínuo do DNA. Este trecho é conhecido como *gene*. A definição do que é um gene gera muita polêmica entre os especialistas da área, e diferentes respostas a este respeito são obtidas quando este assunto é colocado em questão. Como alguns genes codificam apenas moléculas de RNA, para o propósito deste material introdutório, podemos dizer que um gene é um trecho contínuo da molécula de DNA que contém as informações necessárias para construir uma molécula de proteína ou uma molécula de RNA. Esta informação será retirada de apenas uma das tiras de DNA, esta tira é chamada de "template" ou *modelo*.

Um gene completo pode ser bem grande, chegando a conter cerca de 90.000 pb, destes apenas 1,5 % de pares de base são destinados à codificação de aminoácidos. A parte não codificante inclui algumas seqüências que controlam a atividade do gene, mas não se sabe ao certo o

quanto do gene está envolvido neste processo regulatório. Apesar de a informação estar contida no DNA, a síntese de proteína não ocorre diretamente nele. Na verdade, o processo é intermediado por uma molécula de RNA, ou sendo mais específico uma molécula de *RNA mensageiro* ou RNAm. O processo de transferência da informação do DNA para o RNA é chamado de *transcrição*. Neste processo, a fita de RNA é formada sendo complementar à fita modelo do DNA – lembrando que a base complementar a adenina 'A' do DNA será a uracila 'U' no RNA e não a timina 'T' (Figura 10, Figura 11).

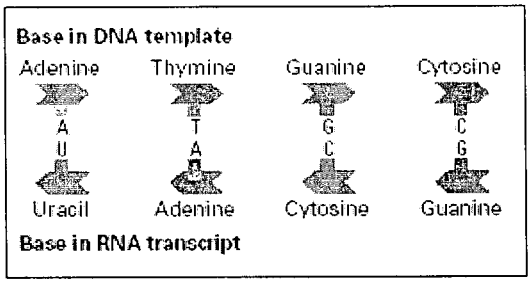

Figura 10 - Transcrição - Bases Complementares [13]

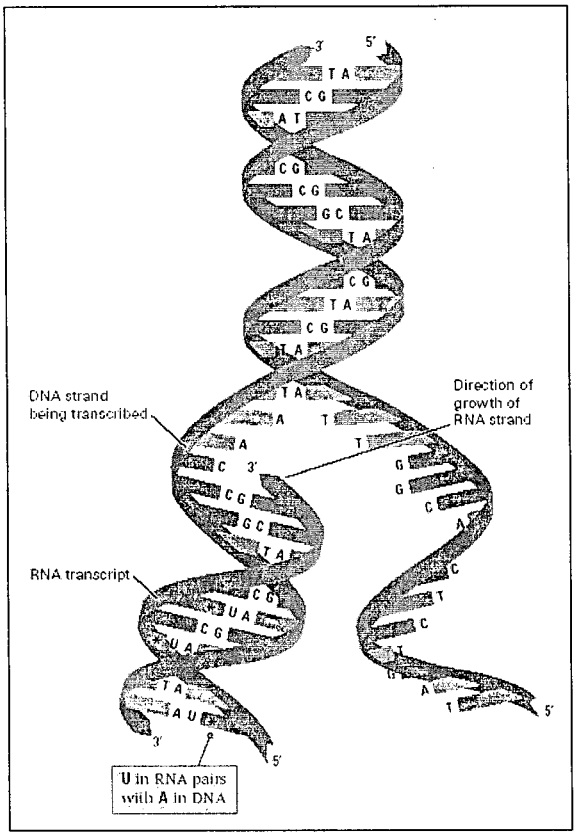

Figura 11- Transcrição – Montagem do RNAm [13]

A partir da molécula de RNAm ocorre um segundo processo chamado de *tradução* onde efetivamente a proteína será construída. O RNAm é traduzido seqüencialmente de três em três nucleotídeos. Estes tripletes são denominados *códons*. Para cada códon no RNAm existe um aminoácido apropriado, que é estabelecido através de um *código genético*. Na natureza existem 20 tipos de aminoácidos. Cada aminoácido é composto por três bases. Como existem 4 tipos de base, teoricamente poderiam haver $4^3 = 64$ aminoácidos. Ocorre que muitas combinações são redundantes, por exemplo, UCU, UCC, UCA, UCG, AGU, e AGC remetem ao mesmo aminoácido *serina*. A Tabela 2, apresenta o código genético. A coluna a esquerda representa o primeiro nucleotídeo do códon, a coluna do meio o segundo e a coluna da direita o terceiro.

Tabela 2 - Código Genético

1ª Posição	2ª Posição								3ª Posição
	U		C		A		G		
U	UUU	Phe(F)	UCU	Ser(S)	UAU	Tyr(Y)	UGU	Cys(C)	U
	UUC		UCC		UAC		UGC		C
	UUA	Leu(L)	UCA		UAA	**TERM**	UGA	**TERM**	A
	UUG		UCG		UAG		UGG	Trp(W)	G
C	CUU	Leu(L)	CCU	Pro(P)	CAU	His(H)	CGU	Arg(R)	U
	CUC		CCC		CAC		CGC		C
	CUA		CCA		CAA	Gln(Q)	CGA		A
	CUG		CCG		CAG		CGG		G
A	AUU	Ile(I)	ACU	Thr(T)	AAU	Asn(N)	AGU	Ser(S)	U
	AUC		ACC		AAC		AGC		C
	AUA		ACA		AAA	Lys(K)	AGA	Arg(R)	A
	AUG	Met(M)	ACG		AAG		AGG		G
G	GUU	Val(V)	GCU	Ala(A)	GAU	Asp(D)	GGU	Gly(G)	U
	GUC		GCC		GAC		GGC		C
	GUA		GCA		GAA	Glu(E)	GGA		A
	GUG		GCG		GAG		GGG		G

O processo de leitura do RNAm é estabelecimento do aminoácido correspondente continua até que um códon remeta a uma sinalização de parada, como é o caso do UAA, UAG e UGA. Após isto, o processo é encerrado e a proteína está formada (Figura 12).

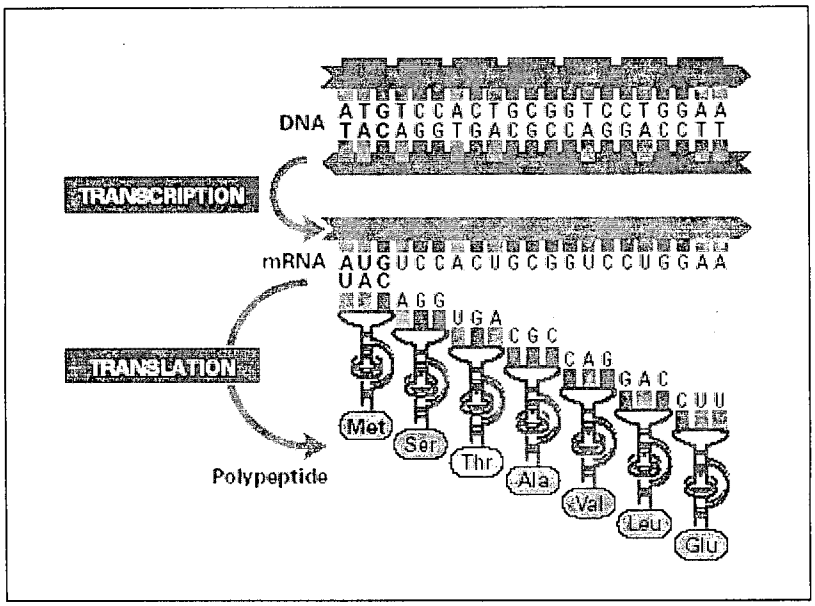

Figura 12 - Tradução e Montagem da Proteína [13]

Este processo de transferência da informação em um único sentido do DNA para o RNA e do RNA para a proteína é denominado *O Dogma Central da Biologia Molecular* (Figura 13). Assim, a transferência de informação do DNA para a proteína é possível, porém da proteína para proteína ou da proteína para o DNA não é possível [14]. O termo dogma remete a certas 'crenças' que foram estabelecidas quando a idéia foi primeiramente posta como teoria. Desde então, diversos experimentos confirmaram a teoria, mas o termo persistiu.

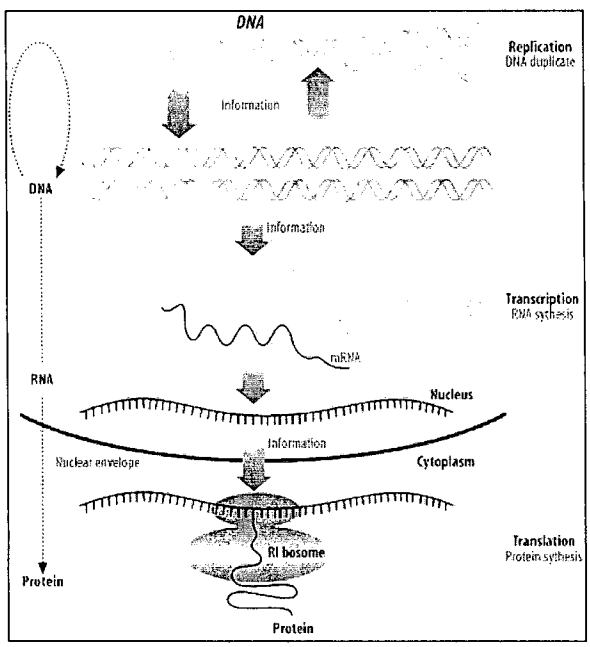

Figura 13 - O Dogma Central da Biologia Molecular [15]

Como um último termo a ser definido, temos que o DNA completo de um organismo é denominado seu *genoma*. Os genomas são por vezes denominados *livros da vida*, principalmente em se tratando do genoma humano. Ler e compreender os diversos livros da vida é um dos principais desafios da era genômica. Tanto a medicina moderna quanto a agricultura e indústria irão cada vez mais depender de um conhecimento íntimo dos genomas para assim poderem desenvolver remédios mais eficientes, selecionar e modificar determinadas características dos organismos bem como compreender melhor os relacionamentos existentes entre as diversas espécies.

Na próxima seção iremos ver uma das principais formas que esta informação contida no DNA pode ser analisada, que é a busca por seqüências similares em bases de dados genômicos, com o uso do algoritmo BLAST e suas implementações computacionais.

2.2 Comparação e Análise de Seqüencias com o Programa BLAST

A comparação de seqüências biomoleculares é uma das tarefas mais importantes da biologia molecular. Esta tarefa consiste basicamente em determinar o quanto duas ou mais seqüências são similares entre si. Não existe uma definição única, precisa e universalmente aplicável à noção de similaridade. Para exemplificar melhor; a seguinte analogia será feita: considerando as palavras 'parte', 'marte' e 'mártir'. Podemos dizer que a pronuncia é bastante similar. Podemos afirmar também que possuem alguma semelhança na forma que são escritas. Porém, não existe nenhuma semelhança no aspecto semântico. Esta situação é um pouco mais regular na biologia molecular: proteínas e DNAs podem ser similares no que diz respeito a sua função, sua estrutura ou sua seqüência primária de aminoácidos e ácidos nucléicos. A regra geral é que a seqüência determina a forma, e a forma determina a função. Assim quando estudamos similaridades entre seqüências, estamos interessados em validar ou descobrir similaridades na forma e na função. Esta abordagem em geral é bem sucedida. Contudo existem casos em que apesar de duas seqüências compartilharem poucas ou quase nenhuma similaridade, as mesmas possuem praticamente a mesma forma e desempenham a mesma função. Nesta dissertação, não são tratados os aspectos relativos à forma e à função das seqüências e assim, estas serão consideradas apenas como cadeias de caracteres.

A *similaridade* possui tanto um aspecto quantitativo quanto qualitativo. O *valor da similaridade* fornece uma resposta quantitativa, informando o *grau de similaridade* que duas seqüências possuem. Já, o aspecto qualitativo pode ser dado com a noção de alinhamento [16]. Um *alinhamento* é um arranjo mutuo entre duas seqüências, no qual fica explícito, onde as mesmas são similares e onde diferem. Um alinhamento pode ser obtido simplesmente emparelhando as duas seqüências. Para obter o melhor alinhamento, é necessário ter uma *função de custo ou pontuação*, onde são atribuídos pontos para cada casamento realizado

entre os caracteres de ambas as seqüências. Os arranjos que obtiverem a maior pontuação serão considerados os alinhamentos ótimos. Como duas seqüências não precisam ser do mesmo tamanho para serem alinhadas, pode haver a necessidade de se inserir *buracos* ou *lacunas* em pontos arbitrários de modo que elas fiquem do mesmo tamanho. Estes buracos também fazem parte do modelo de custo e recebem uma pontuação negativa. Na Figura 14 é apresentado um alinhamento entre duas seqüências, com pontuação ou grau de similaridade '5'. Neste exemplo é utilizada uma função de custo, que atribuí o valor '1' para casamentos exatos, '-1' para casamentos mal sucedidos e '-2' para buracos inseridos. Os buracos são representados por um traço '-'.

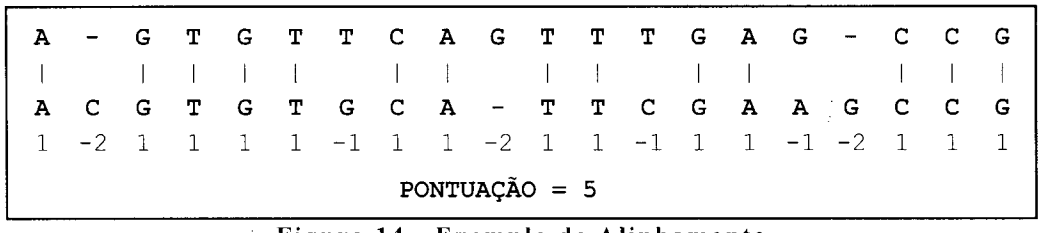

Figura 14 - Exemplo de Alinhamento

Existem diversas formas de se alinhar duas seqüências. O uso de computadores é largamente utilizado nesta tarefa e diversos algoritmos foram propostos para resolver este problema. Encontrar os alinhamentos ótimos pode ser uma tarefa computacionalmente complicada. Porém, métodos de *programação dinâmica* tornaram o problema tratável. Existem basicamente dois tipos de algoritmos para alinhamentos: os que geram *alinhamentos globais* e os que geram *alinhamentos locais*. Em um alinhamento global deseja-se emparelhar duas seqüências **S** e **T** de tal forma que se obtenha o melhor alinhamento possível entre as duas. Porém, em algumas situações, duas seqüências podem não ser muito similares como um todo, mas podem conter subseqüências – **a** de **S** e **b** de **T** – que o sejam. Esta é a noção de alinhamento local. Fazendo uma analogia, se as seqüências a serem alinhadas fossem duas sentenças em português, teríamos que o alinhamento global seria a melhor forma de se emparelhar as mesmas. Já os alinhamentos locais seriam trechos

emparelhados de ambas as sentenças que fossem semelhantes (Figura 15).

```
Sentenças:

|----+----1----+----2----+----3----+----4----+----5----+----6

"Eu fui fazer compras de mercado e voltei cansado."
"Luciano guardou as compras de mercado e ficou muito cansado."

Alinhamento Global:

"------Eu fui fazer compras de mercado e voltei cansado.-----"
"Luciano guardou as compras de mercado e ficou muito cansado."

Alinhamentos Locais:

Sentença 1: 13 " compras de mercado " 32
Sentença 2: 19 " compras de mercado " 38

Sentença 1: 41 " cansado." 49
Sentença 2: 52 " cansado." 60
```

Figura 15 - Alinhamento Global x Local

Os dois algoritmos mais utilizados para obtenção de alinhamentos globais e locais são os de Needdleman-Wunsch [17] e Smith-Waterman [18], respectivamente. Nesses métodos, para determinarmos a pontuação de cada alinhamento, definimos um espaço de busca entre as duas seqüências que pode ser visualizado como um gráfico em que uma das seqüências é posicionada ao longo do eixo X e a outra seqüência é posicionada ao longo do eixo Y. Cada ponto presente neste espaço representa um casamento entre duas letras, uma de cada seqüência. Cada par de letras recebe uma pontuação vinda de uma matriz – função de custo – cujos valores foram determinados empiricamente (BLOSUM62, PAM250). Neste quadro, pares de segmentos ou alinhamentos sem lacunas aparecem como linhas diagonais no espaço de busca, e a pontuação destes alinhamentos é simplesmente a soma das pontuações de cada par de letras que foi casado. Já os alinhamentos que contém lacunas aparecem como diagonais quebradas, e a sua pontuação é a soma dos pares de letras menos a soma das lacunas existentes. Por questões biológicas, as lacunas recebem uma penalidade maior quando se iniciam do que quando se estendem (Figura 16).

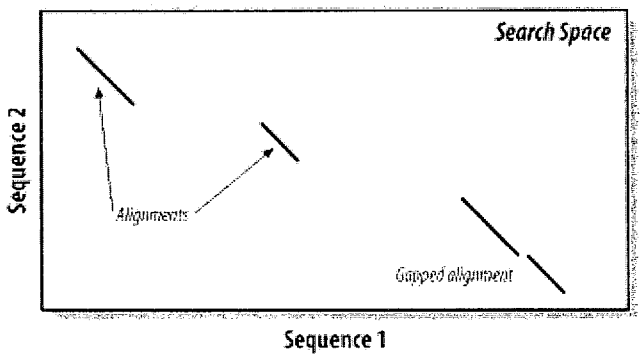

Figura 16 - Espaço de Busca e Alinhamentos [15]

A complexidade de tempo desses algoritmos deriva do tamanho do espaço de busca. Como nestes algoritmos uma pontuação é atribuída para cada ponto no espaço de busca, a complexidade de tempo para obter-se a melhor pontuação é quadrática, da ordem de **O(nm)**, onde **n** e **m** são os tamanhos da duas seqüências. Assim, para o alinhamento entre duas seqüências, estes algoritmos apresentam um desempenho aceitável. Porém, quando desejamos comparar, uma ou mais seqüências, denominadas *seqüências de consulta*, com uma base de dados com **k** seqüências, estes algoritmos passam a ter complexidade proporcional ao tamanho (em caracteres) total da base de dados, vezes o tamanho (em caracteres) total das seqüências de consulta. Assim, para estes casos, a complexidades destes algoritmos se torna proibitiva, e métodos alternativos que utilizam heurísticas foram desenvolvidos.

Assim a família de algoritmos BLAST [2, 3] e FASTA [19] são baseados em métodos heurísticos que reduzem o tempo computacional, diminuindo de alguma forma a sensibilidade do algoritmo. Estes métodos reduzem o tamanho do problema, selecionando da base de dados somente seqüências que tenham uma maior probabilidade de possuírem alguma semelhança com a seqüência de entrada; procurando dentro destas as regiões com similaridade. Esta etapa de seleção permite confinar o custoso tempo de alinhamento, somente em um subconjunto das seqüências armazenadas no banco, além de restringir a busca aos melhores alinhamentos a somente um trecho destas seqüências. Para serem rápidos estes algoritmos estimam a similaridade entre as seqüências de maneira aproximada e, portanto, introduzem um risco de

se desprezar seqüências cuja similaridade seja mais sutil e difícil de se detectar. Os detalhes de como o BLAST realiza tal pesquisa não são tão simples, consistindo basicamente de três passos principais, que serão descritos na próxima seção.

2.2.1 O BLAST

A família de programas BLAST nasceu em 1990, com o desenvolvimento do BLAST1 [3], que era bastante rápido e dedicava-se a busca de similaridades através de alinhamentos locais sem buracos. A motivação para o desenvolvimento do BLAST foi a necessidade de aumentar a velocidade de execução do algoritmo FASTA. Tal objetivo foi alcançado, escolhendo-se menos e melhores pontos iniciais de busca.

Em 1996-1997, duas novas versões do BLAST, levemente diferentes, surgiram. Ambas diferenciam-se do BLAST original por permitirem buracos nos alinhamentos. Estas versões foram denominadas por seus autores por BLAST2, e passaram a ser distinguidas uma da outra por carregarem em suas iniciais o nome das instituições que as desenvolveram: NCBI-BLAST "National Center for Biotechnology Information" [5] e WU-BLAST "Washington University" [20].

A execução do BLAST é iniciada informando ao programa, uma ou mais seqüências alvo, também chamadas seqüências de consulta; uma ou mais bases de dados de seqüências, que contém um conjunto de seqüências já conhecidas, seqüências essas que serão alinhadas com as seqüências de consulta na busca por alinhamentos locais com elevado grau de similaridade; além de uma série de outros parâmetros a serem utilizados na busca, que serão detalhados no decorrer desta seção.

O BLAST foi desenvolvido inicialmente para busca de alinhamentos em seqüências de aminoácidos, porém seu método foi estendido para também efetuar alinhamentos em seqüências de

nucleotídeos. Assim, de acordo com a natureza (nucleotídeos ou aminoácidos) da base de dados e da consulta, um tipo de BLAST deverá ser executado. Existem cinco tipos ou 'sabores' de pesquisa BLAST (Tabela 3).

Tabela 3 - Os Sabores do BLAST

Programa	Base de Dados	Consulta
BLASTN	Nucleotídeos	Nucleotídeos
BLASTP	Aminoácidos	Aminoácidos
BLASTX	Aminoácidos	Nucleotídeos traduzidos em aminoácidos
TBLASTN	Nucleotídeos traduzidos em aminoácidos	Aminoácidos
TBLASTX	Nucleotídeos traduzidos em aminoácidos	Nucleotídeos traduzidos em aminoácidos

A menos das transformações inicias da consulta ou da base de dados e de algumas restrições a determinados parâmetros, o algoritmo do BLAST é praticamente o mesmo para qualquer tipo de busca. Antes de iniciar a descrição de como o BLAST efetua essa busca alguns termos serão definidos:

- *segmento*: subseqüência contígua de uma seqüência de nucleotídeos ou aminoácidos;
- *par de segmentos* ou *alinhamento local*: um par de segmentos alinhados (podendo conter lacunas) de mesmo tamanho vindo das duas seqüências que estão sendo comparadas.
- *par de segmentos de alta pontuação*, "high-scoring segment pair" ou HSP: alinhamento local, cuja pontuação ultrapasse um patamar determinado.

Dado estas definições, podemos dizer que o objetivo do BLAST é procurar pelos HSPs existentes entre as seqüências de consulta e as seqüências da base de dados. Deve-se ficar claro que o objetivo do

BLAST é encontrar alinhamentos locais de alta pontuação – HSPs – e não alinhamentos globais, cujo objetivo por sua vez seria alinhar duas seqüências como um todo.

Como já visto, alguns algoritmos existentes estão preocupados em encontrar a máxima pontuação nos alinhamentos locais existentes entre duas seqüências, como é o caso do algoritmo de Smith-Waterman [18]. Porém, o que o BLAST faz é determinar todos os alinhamentos locais, que sejam estatisticamente relevantes. Contudo, diferentemente do algoritmo Smith-Waterman, o BLAST não explora todo o espaço de busca para determiná-los. Esta minimização do espaço de busca que o BLAST realiza é o ponto chave da sua rapidez, mesmo que para isso tenha uma perda de sensibilidade. Assim existe uma relação velocidade versus sensibilidade que é um conceito bastante importante quando realizamos experimentos com o BLAST. Para evitar percorrer todo o espaço de busca, o BLAST utiliza heurísticas implementadas em três passos principais que seqüencialmente refinam potenciais HSPs. Estes passos, descritos nas próximas seções, são conhecidos como escolha das sementes "seeding", extensão das sementes "extension", e avaliação dos alinhamentos "evaluation" [15, 21].

2.2.1.1 A Escolha das Sementes

A primeira heurística utilizada pelo BLAST é de que alinhamentos significativos possuem pequenas palavras em comum. Uma palavra "w-mers" (mers é derivado da palavra "polymer") é apenas um número determinado de letras em seqüência. Por exemplo, se for definido uma palavra como sendo de três letras, 3-mers, então teríamos que a seqüência 'SVIGFR' possui as palavras: 'SVI', 'VIG', 'IGF' e 'GFR' (Figura 17).

```
SVIGFR
SVI
 VIG
  IGF
   GFR
```

Figura 17 - Palavras de uma Seqüência

Quando o BLAST compara duas seqüências, ele primeiramente determina a localização de todas as palavras em comum, que são denominados acertos ou "word hits" (Figura 18). Somente estas regiões é que serão utilizadas como pontos de partida para busca de HSPs. Desta forma, O BLAST ignora uma boa parte do espaço de busca.

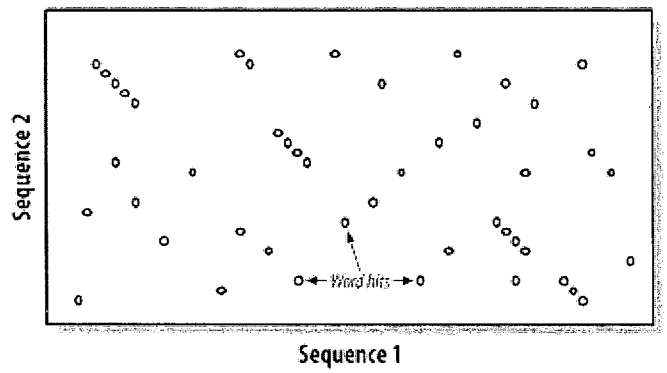

Figura 18 - Pontos de acerto [15]

A definição do que é um acerto até aqui foi um pouco simplificada, dando a impressão de que o mesmo significa a presença de duas palavras idênticas. Porém muitos alinhamentos significativos não possuem nenhuma palavra idêntica em comum. Assim o BLAST emprega um conceito um pouco mais preciso de acerto que é por ele denominado de vizinhança "neighborhood". A vizinhança de uma palavra é uma lista que contém a própria palavra além de todas as outras palavras cuja pontuação é no mínimo **T** quando comparada com a mesma via uma matriz de pontuação. Portanto ajustando o valor de **T**, é possível controlar o tamanho da lista de palavras vizinhas e por conseqüência também controlar o número de acertos no espaço de busca. A Tabela 4 mostra as palavras vizinhas à palavra RGD, com a pontuação sendo dada a partir de duas matrizes de pontuação diferentes, a BLOSUM62 e a PAM200.

Tabela 4 - Lista de Palavras Vizinhas

BLOSUM62		PAM200	
Palavra	Pontuação	Palavra	Pontuação
RGD	17	RGD	18
KGD	14	RGE	17
QGD	13	RGN	16
RGE	13	KGD	15
EGD	12	RGQ	15
HGD	12	KGE	14
NGD	12	HGD	13
RGN	12	KGN	13
AGD	11	RAD	13
MGD	11	RGA	13
RAD	11	RGG	13
RGQ	11	RGH	13
RGS	11	RGK	13
RND	11	RGS	13
RSD	11	RGT	13
SGD	11	RSD	13
TGD	11	WGD	13

Um valor apropriado para **T** irá depender tanto dos valores da matriz de pontuação, quanto da relação velocidade e sensibilidade. Altos valores para **T** progressivamente removem o número de acertos e reduzem o espaço de busca. Isto pode fazer o BLAST ser executado mais rapidamente, porém, a chance de se perder um alinhamento relevante aumenta (Figura 19).

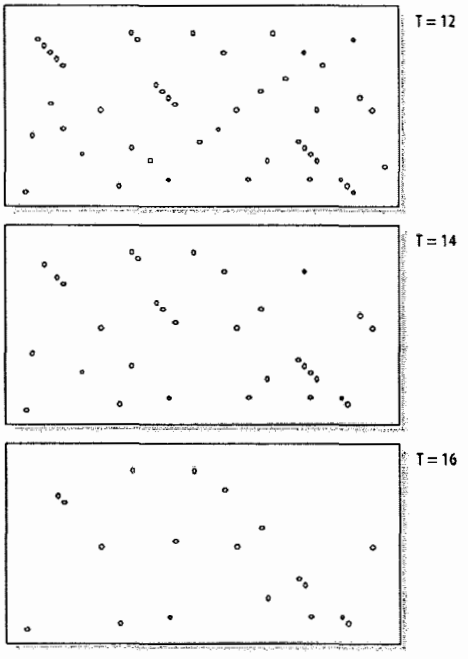

Figura 19 - Redução do espaço de busca conforme aumento de T [15]

O tamanho da palavra **W** é uma outra variável que controla o número de acertos. Fica fácil verificar que para **W=1** o número de acertos será bem maior do que para **W=10**. Assim há uma relação entre **T**, **W** e a matriz de pontuação, de forma que um ajuste fino destes parâmetros é a maneira ideal para se controlar a velocidade e a sensibilidade na execução do BLAST.

Uma característica observada que se tornou uma importante heurística nas versões posteriores do BLAST é de que os acertos tendem a se agrupar em torno de diagonais no espaço de busca (Figura 20). O *algoritmo dos dois-acertos* "two-hit algorithm" toma vantagem desta característica, requerendo que dois acertos estejam em uma mesma diagonal a uma distância máxima **A** um do outro, para que assim sejam candidatos a etapa posterior de extensão. Esta distância **A** é denominada *tamanho da janela para múltiplos acertos* "multiple hits window size". Estabelecer um valor pequeno para **A** tende a reduzir o espaço de busca.

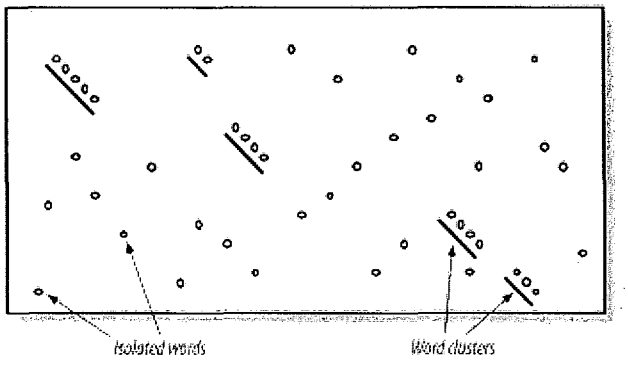

Figura 20 - Agrupamento de acertos em torno de diagonais [15]

Uma observação a ser feita é que de maneira diferente do que ocorre na execução do BLAST para alinhamento de proteínas (BLASTp), quando o BLAST é executado para alinhamento de nucleotídeos (BLASTn) não ocorre a construção da lista de palavras vizinhas e, portanto são escolhidas como sementes apenas palavras idênticas. Assim, o parâmetro **T** nunca é utilizado e, portanto, a única forma de se fazer o BLASTn executar mais rapidamente é aumentando o tamanho da palavra, ou seja, o valor de **W**. Outra observação importante é de que o algoritmo dos dois-acertos também não é utilizado na execução do BLASTn devido

ao fato de que acertos são mais raros quando se busca palavras de tamanho grande e idênticas.

A versão do BLAST disponibilizada pelo NCBI utiliza no BLASTn o valor **11** para **W** como padrão, já no BLASTp os valores padrões para **W**, **T** e **A** são respectivamente **3**, **11** e **40** [5, 20].

2.2.1.2 A Extensão dos Acertos

Uma vez que os pontos de partida (acertos) tenham sido definidos no espaço de busca, os alinhamentos podem ser gerados a partir dos mesmos. No algoritmo Smith-Waterman [18], as extremidades do melhor alinhamento são determinadas somente após todo o espaço de busca ter sido analisado. Contudo como o BLAST reduz o espaço de busca, ele precisa ter um mecanismo para saber quando parar o processo de extensão (Figura 21).

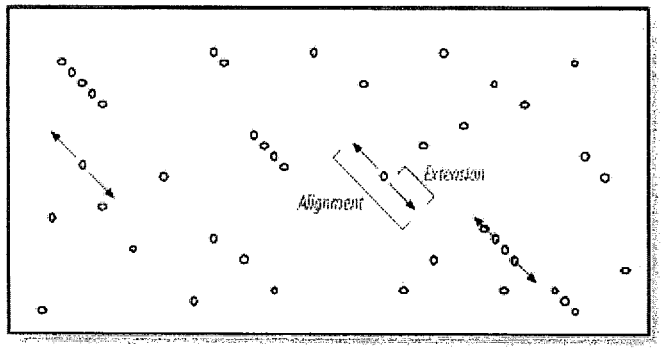

Figura 21 - Extensão dos Acertos [15]

Para exemplificar este processo poderíamos tentar alinhar as duas seqüências a seguir (Figura 22). Por simplicidade será assumido que para acertos a pontuação será **+1** e para uma combinação mal sucedida a pontuação **-1**:

```
----+----1----+----2----+----3----+----4
AGCTTTTCATTCTTCACAGCTGCATTAGGTCAGGC
AGCTTTATATTCTATCAACTTCAAGAAGTCAGACA
```

Figura 22 - Seqüências a serem Alinhadas

Assumindo que o ponto de partida utilizado seja a letra **A** do início da seqüência, e que a extensão será para a direita, o alinhamento ocorreria sem nenhuma falha até a sétima letra de cada seqüência (Figura 23).

```
    ----+----1
    AGCTTTT
    AGCTTTA
```

Figura 23 - Primeiro Erro

Aqui uma decisão precisará ser tomada para se saber se o alinhamento deveria continuar ou não. Se fosse possível olhar um pouco para frente ficaria claro que se deveria continuar, mesmo que o final de ambas as seqüências sejam bastante diferentes, o que levará a muitos erros. Para permitir tomar esta decisão, uma variável **X** é estabelecida de forma a representar o quanto é permitido à pontuação do alinhamento decair em relação à pontuação máxima obtida até então. Caso **X** atinja o valor estabelecido, o alinhamento é interrompido. Assim para **X=5** o alinhamento se comportaria conforme a Figura 24.

```
    ----+----1----+----2----+----3----+----4
    AGCTTTTCATTCTTCACAGC
    AGCTTTATATTCTATCAACT
    12345654567898765654  <- Pontuação
    12345666667899999999  <- Pontuação Máxima
    00000012100001234345  <- Histórico de X
```

Figura 24 - Evolução do alinhamento para X=5

A pontuação do alinhamento ao atingir o ponto máximo de queda, **X=5**, faz com que o processo seja interrompido, e o alinhamento é retornado, no caso, do início até a última posição onde a maior pontuação, nove, foi obtida (Figura 25).

```
    ----+----1----+----2
    AGCTTTTCATTCT
    AGCTTTATATTCT
```

Figura 25 - Alinhamento Retornado

Deve-se notar que o valor escolhido para **X** pode acarretar num término prematuro de um alinhamento. No exemplo anterior caso fosse estabelecido **X=2**, teríamos um alinhamento (Figura 26) de pontuação menor, no caso seis.

```
----+----1----+----2----+----3----+----4
AGCTTTTC
AGCTTTAT
12345654      <- Pontuação
00000012      <- Histórico de X
12345666      <- Pontuação Máxima.
```

Figura 26 - Evolução do alinhamento para X=2

Caso fosse estabelecido **X=3**, teríamos um alinhamento tão bom quanto **X=5** (Figura 27).

```
----+----1----+----2----+----3----+----4
AGCTTTTCATTCTTCA
AGCTTTATATTCTTCA
1234565456789876 <- Pontuação
0000001210000123 <- Histórico de X
123456    789    <- Pontuação Máxima.
```

Figura 27 - Evolução do alinhamento para X=3

Portanto conforme observado, atribuir valores pequenos para **X** pode interromper prematuramente um alinhamento, já atribuir valores elevados pode acarretar em um volume de processamento desnecessário, visto que a pontuação máxima já terá sido atingida muito antes de **X** atingir o valor limite. Na versão do NCBI o valor padrão de **X** para o BLASTp e o BLASTn são respectivamente **7** e **20**, quando não se permite inserção de lacunas, versão "ungapped" e, **15** e **30** caso contrário (versão padrão).

2.2.1.3 A Etapa de Avaliação

Uma vez que os acertos determinados na primeira etapa do algoritmo tenham sido expandidos em ambas as direções, estes são então avaliados para determinar se são estatisticamente significantes. Aqueles

que forem significantes serão denominados HSPs. De uma forma simplificada, avaliar um alinhamento não é complicado, basta estabelecer um limite **S** relativo à pontuação dos alinhamentos, para qualificá-los como sendo de alta ou baixa pontuação. Esta variável **S** está diretamente relacionada a uma outra variável **E** através da equação de Karlin-Altschul [22].

$$E = kmne^{-\lambda S} \quad \textbf{Eq. 1}$$

E representa o número de alinhamentos que se espera encontrar ao acaso. Esta equação estabelece que **E** é determinado em função do tamanho do espaço de busca **(m*n)**, a pontuação normalizada **(λS)** e a constante **k**. Devido a esta relação, **S** também é denominadao de limite estatístico. Este limite **S** é uma maneira bastante eficiente para se remover alinhamentos aleatórios ou de baixa pontuação. Porém, se seu valor for excessivamente elevado, nenhum alinhamento poderá ser retornado (Figura 28).

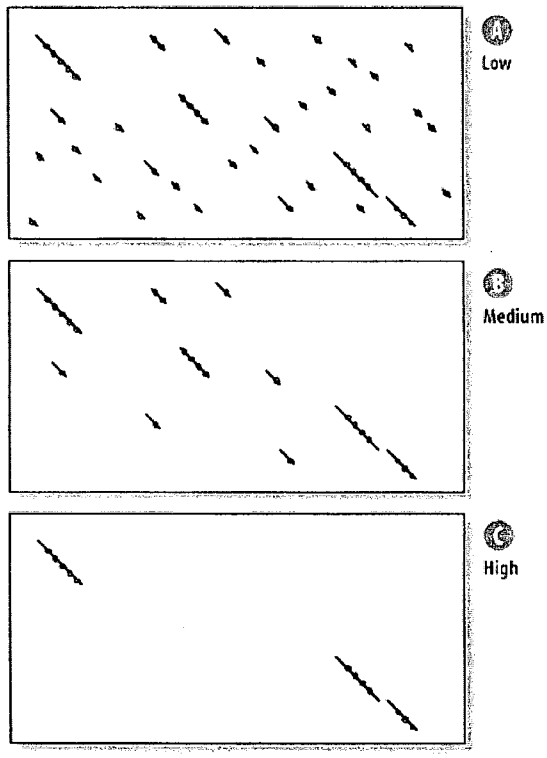

Figura 28 - Relação entre S e o número de alinhamentos Retornados [15]

Outro aspecto a ser observado é que o relacionamento entre os HSPs selecionados deve ser baseado no mesmo relacionamento existente entre alinhamentos sem lacunas, ou seja, as linhas no gráfico devem partir do canto superior esquerdo em direção ao canto inferior direito, não devendo haver sobreposição de coordenadas. Os grupos de HSPs que se comportam desta forma são ditos consistentes (Figura 29).

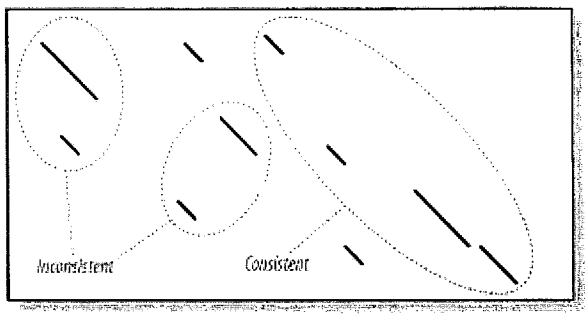

Figura 29 - Grupos de alinhamentos consistentes e inconsistentes [15]

O algoritmo para definição de grupos de HSPs consistentes, compara as coordenadas de todos os HSPs para verificar se há sobreposição. Esta operação tem complexidade quadrática em relação ao número de HSPs e portanto pode vir a ser custosa, dependendo da seletividade da consulta. Uma vez determinados, os HSPs são retornados pelo BLAST, em um formato específico (Figura 30).

```
TBLASTX 2.2.10 [Oct-19-2004]

Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer,
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997),
"Gapped BLAST and PSI-BLAST: a new generation of protein database search
programs",  Nucleic Acids Res. 25:3389-3402.

Query= 7835
         (114 letters)

Database: nt.small
           574,855 sequences; 2,055,228,948 total letters

                                                                    Score    E
Sequences producing significant alignments:                         (bits) Value

emb|AL662893.12| Mouse DNA sequence from clone RP23-202G21 on ch...    32   2.9
dbj|AP006441.3| Oryza sativa (japonica cultivar-group) genomic D...    31   5.5

>emb|AL662893.12| Mouse DNA sequence from clone RP23-202G21 on chromosome 11, complete
              sequence
          Length = 205405

 Score = 31.8 bits (63), Expect = 2.9
 Identities = 12/35 (34%), Positives = 21/35 (60%)
 Frame = +1 / -3

Query: 1      CI*WILSLYLVI*LLEHQDNYIYLSNVAFLLVIQL 105
               C+ WILS Y +  +L H   ++++  +AF  V +L
Sbjct: 194051 CLFWILSFYQMEKILSHSVGFLFIQLIAFWAVQKL 193947

>dbj|AP006441.3| Oryza sativa (japonica cultivar-group) genomic DNA, chromosome 9, BAC
             clone:B1279D09
          Length = 134793

 Score = 30.9 bits (61), Expect = 5.5
 Identities = 12/32 (37%), Positives = 19/32 (59%)
 Frame = +1 / +3

Query: 10     WILSLYLVI*LLEHQDNYIYLSNVAFLLVIQL 105
               W L  Y++  L  HQ    YL ++AF+L+ Q+
Sbjct: 17553 WFLLDYIIAFLHYHQSIVYYLQSIAFILLYQI 17648

  Database: nt.small
    Posted date:  Jun 21, 2005  1:40 PM
  Number of letters in database: 2,055,228,948
  Number of sequences in database:  574,855

Lambda     K      H
   0.318    0.134    0.401

Matrix: BLOSUM62
Number of Hits to DB: 470,001,508
Number of Sequences: 574855
Number of extensions: 3114933
Number of successful extensions: 88914
Number of sequences better than 10.0: 4
length of database: 685,076,316
effective HSP length: 29
effective length of database: 668,405,521
effective search space used: 5347244168
frameshift window, decay const: 40,  0.5
T: 13
A: 40
X1: 16 ( 7.3 bits)
X2: 0 ( 0.0 bits)
S1: 41 (21.7 bits)
```

Figura 30 - Resultado do BLAST

2.2.2 Bases de Dados do BLAST

Conforme já foi mencionado, para efetuar um busca BLAST é necessário fornecer diversos parâmetros, entre eles a base de dados a ser consultada. Estas bases de dados possuem, em geral, seqüências previamente estudadas, que poderão fornecer alguma informação a respeito das seqüências de consulta através das similaridades encontradas. Mas o que seriam estas bases de dados e de onde estas são obtidas?

As bases de dados utilizadas pelo BLAST, são arquivos texto utilizados como repositórios de dados, onde uma ou mais seqüências são armazenadas, cada uma contendo uma identificação – se possível única – além dos dados da seqüência em si, ou seja, sua cadeia de aminoácidos ou nucleotídeos. Na maioria dos casos, as buscas BLAST são efetuadas utilizando bases de dados públicas. As seqüências presentes nas bases de dados públicas são extraídas de alguns repositórios internacionais de dados. Tais repositórios possuem diversas informações a respeito de cada seqüência, que vão além da seqüência em si e sua identificação. Neles encontramos informações do tipo: origem da seqüência, função, etc. Apesar de não haver um único repositório onde se encontre qualquer informação necessária a esse respeito, existe um repositório em especial que é o "International Nucleotide Sequence Database Collaboration" ou INSD. O INSD é um consórcio que tem por objetivo manter o maior repositório público de seqüências de DNA e proteína do mundo. Ele é composto por três partes principais:

- O "DNA Data Bank of Japan" ou DDBJ (http://www.ddbj.nig.ac.jp);
- O "European Molecular Biology Laboratory" ou EMBL (http://www.embl.org);
- O GenBank do "National Center for Biotechnology Information" ou NCBI (http://ncbi.nlm.nih.gov/GenBank).

Outros repositórios públicos também importantes seriam:

- SWISS-PROT: "Protein Knowledgebase" (http://www.ebi.ac.uk/swissprot)
- TrEMBL: (http://www.ebi.ac.uk/tremble)
- UniGene: (http://www.ncbi.nlm.nih.gov/UniGene)

É importante notar que, apesar de muitos destes repositórios públicos serem referenciados como bancos de dados, estes em geral não utilizam sistemas gerenciadores de banco de dados (SGBD) propriamente ditos, mas sim sistemas de arquivos do sistema operacional ou sistemas de arquivos próprios, ou uma combinação dos três. Para efeitos de esclarecimento, um SGBD seria uma coleção de programas que permite criar e manter uma base de dados. Uma base de dados por sua vez seria uma coleção lógica e coerente de dados, com algum significado agregado. Assim, um SGBD prove meios de definir, construir, e manipular os dados de uma base de dados, de forma eficiente, consistente, controlada e segura [23].

Para ser mais preciso – no contexto do BLAST – com o termo base de dados e para facilitar a compreensão de como as seqüências biológicas são armazenadas, este tópico irá dar uma breve descrição dos formatos de arquivo suportados pelo BLAST.

2.2.2.1 O Formato FASTA

O *formato FASTA* é um dos formatos padronizados para armazenamento de dados de seqüências biomoleculares. É um formato bastante simples, que consiste de duas partes: a linha de identificação e as linhas da seqüência propriamente dita.

A parte de identificação (Figura 31) segue a seguinte regra de formação: um sinal de maior '>' seguido de um identificador da seqüência – que não pode conter espaços em branco – seguido opcionalmente de um espaço em branco e a descrição da seqüência. Em geral esta descrição se encontra somente na primeira linha, porém caso seja necessário, ela pode ser continuada nas linhas seguintes, desde que os caracteres utilizados para mudança de linha sejam um 'CTRL+A' ('^A'). Estes caracteres não aparecem na maioria dos editores de texto, mas geram o mesmo efeito dos caracteres de controle 'NL-New Line + CR-Carriage Return'. Tal restrição é necessária, pois o 'NL+CR' é utilizado como delimitador entre a parte da descrição e a parte dos dados da seqüência.

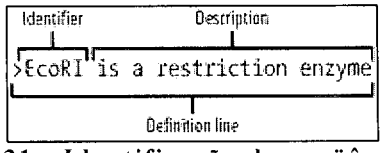

Figura 31 - Identificação da seqüência [15]

Nas linhas seguintes à parte da identificação, seguem os dados da seqüência propriamente dita. Em geral tais dados são dispostos, por convenção, em no máximo 80 caracteres por linha, mas isto não é obrigatório. Podem ser utilizadas quantas linhas forem necessárias, e o seu fim ocorre ou com o fim do arquivo, ou com o início da identificação de uma nova seqüência (Figura 32).

```
>gi|2367165|gb|AE000368.1|AE000368 Escherichia coli K-12 MG1655 section 258 of 400 of the
complete genome
TGAATAAGGAACGAGACAAACGCCTCAACGGCCAAGTGCCCAATCTCTATTAACGAAAAAAGGGCCGGATGTACAGCACA
TCCGGCCCGTGAAATCAGACGCCGATATTTCTCAACTTCTCGCCTGCCATCAGTTTGCGTTCGATATGTTCCAGCGTGAC
ATTTTTGGTTTCCGGAATGAGCCAGAAAGTAATGCCCACAAACGCAATGTTCAGCGCAGTGTAGAGCCAGAACGTACCGG
CAG
>gi|1787467|gb|AE000220.1|AE000220 Escherichia coli K-12
AGTGTCTGGTTTCAAGATTAGCCCCCGTTCTGTTGTCAGGTTTTACCTCTCAACGTGCGGGGGTTTTCTCTTTCCAGCAA
CCAATGCCACCAGGGATAAAGCCCCCGCAACATTGCGCCTCACCGGATAATTCCGGCTTGGTGTGGATACTACGTCTCAA
TTCATCTTCACTTCATCCCTGAAATGTTTGCAATAAAGAGTACATTCCGGCTTTTCAACAGCTGTTGCAGTCGTTTCATG
>gi|2367168|gb|AE000369.1|AE000369 Escherichia coli K-12 MG1655 section 259 of 400 of the
complete genome
TAATGTGCCGATAACAAATATAGTCATTCTACGTAACGTCTCCATAAGGTGATATTTGACATTATCAGAAGCTGCGAATT
>gi|1790858|gb|AE000510.1|AE000510 Escherichia coli K-12 MG1655
GACATCAGGTAA
```

Figura 32 - Seqüências no formato FASTA

2.2.2.2 As bases de dados BLAST

As bases de dados utilizadas pelo BLAST são na verdade arquivos texto contendo uma ou mais seqüências no formato FASTA. Estes arquivos devem conter apenas seqüências do mesmo tipo, ou seja, ou de nucleotídeos ou de proteínas.

Quando há mais de uma seqüência de consulta, estas também podem ser agrupadas em um único arquivo, embora cada uma venha a ser submetida individualmente pelo BLAST. Aqui vale também a mesma regra e assim este arquivo deve conter apenas seqüências de um mesmo tipo. Portanto apesar do BLAST utilizar os termos consulta e base de dados, na verdade não há diferença entre estes, ambos são arquivos no formato FASTA, e em um dado momento uma base de dados pode vir a ser utilizada como arquivo de consultas e vice e versa.

Um último detalhe a ser coberto neste assunto seria o termo *base de dados BLAST* ou "BLAST databases". Pela natureza bastante repetitiva dos caracteres que representam ou os nucleotídeos ou as proteínas, um arquivo no formato FASTA é altamente compactável. Baseado nisto o BLAST impõe uma restrição de que quando utilizado como base de dados, o arquivo no formato FASTA tenha que ser formatado segundo um aplicativo chamado *formatdb*. Tal aplicativo tem por função compactar o arquivo no formato FASTA e gerar alguns outros arquivos que facilitem o acesso aos dados. Desta forma o BLAST consegue acelerar em muito o processo de comparação entre as seqüências de consulta e a seqüências da base de dados. Por ser em geral um arquivo relativamente pequeno, o BLAST não exige o mesmo procedimento para o arquivo de consultas, efetuando o mesmo processo – a compactação - em tempo de execução.

Nos capítulos seguintes, o termo base de dados será utilizado de forma livre, sem diferenciar se a mesma é um arquivo no formato FASTA

ou uma base de dados BLAST, tal diferenciação somente será feita nos casos imprescindíveis à compreensão. O termo consulta também será utilizado de maneira livre, sem diferenciar arquivo de consultas de uma única seqüência de consulta, fazendo tal diferenciação somente nas ocasiões necessárias.

2.3 Paralelismo no BLAST

Apesar de ser bastante eficiente, o BLAST vem se deparando com um problema que afeta em muito o seu tempo de resposta: a taxa de crescimento das bases de dados. A base do GenBank, por exemplo, passou a crescer exponencialmente partir de 1994, com cerca de 9 milhões de novas seqüências adicionadas por ano [24] (Figura 33). Atualmente o tamanho de muitas destas bases são da ordem de Giga Bytes, e dado os avanços nas técnicas de seqüênciamento, estas tendem a crescer cada vez mais rápido.

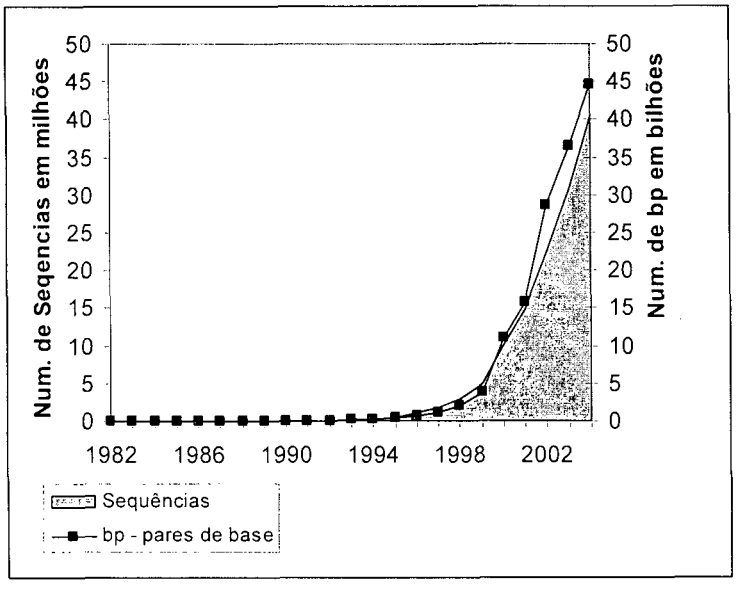

Figura 33 - Crescimento do GenBank. Dados de 02/2005 [25]

Como visto na seção 2.2 , o BLAST é em geral a primeira ferramenta utilizada por um pesquisador quando este obtém uma nova seqüência de proteínas ou nucleotídeos. Ocorre que os dados que hoje fazem parte dos repositórios públicos, são os mesmo que outrora os pesquisadores obtiveram. Assim ao mesmo tempo em que as bases crescem, o volume de consultas BLAST também cresce, na mesma proporção. Devido à forma como o BLAST foi projetado, com acesso a arquivos textos, grande consumo de memória e processamento, este exige e irá exigir cada vez mais um alto poder computacional.

Dado este cenário, é cada vez maior a busca por soluções que possam acelerar a execução do BLAST. Uma delas seria submeter o BLAST a um processo de reengenharia tornando seu código mais adequado às máquinas mais recentes. Atualmente os pacotes de programas que implementam o algoritmo do BLAST possuem uma estrutura muito acoplada, onde o algoritmo em si se mistura com a maneira de se acessar os dados, o que dificulta em muito a manutenção e melhoria do mesmo. Sofrendo o processo de reengenharia, o sistema poderia ser dividido em camadas e módulos bem específicos, separando, por exemplo, a forma de acessar os dados, do algoritmo de alinhamento. Assim, otimizações na camada de acesso aos dados, como políticas de cache, índices, uso de SGBD, etc., poderiam ser feitas sem comprometer o algoritmo e vice-versa. Esta estratégia exigiria um esforço muito grande, visto que a versão disponibilizada atualmente pelo NCBI possui milhares de linhas de código.

Uma outra alternativa seria fazer uso do paralelismo. Basicamente a computação paralela aplica a antiga regra do *dividir para conquistar* para obter maior capacidade de processamento. Nesta estratégia divide-se o problema maior em problemas menores e estes são executados em paralelo. Essa estratégia exige o acesso e a modificação do código fonte do programa a ser paralelizado. Outra estratégia é conhecida como paralelismo de dados. Esse paralelismo pode ser aplicado quando um programa realiza as mesmas operações sobre conjuntos distintos de dados de forma independente. Assim, de certa forma, se mantém o código seqüencial, que é executado em paralelo ao processar diferentes sub-conjuntos de dados.

No caso do BLAST, devido às dificuldades de manipulação do código fonte, essa ferramenta pode ser considerada como uma 'caixa-preta'. Assim, ao invés de realizar uma paralelização no código fonte do BLAST, uma alternativa interessante é a obtenção do paralelismo através da distribuição de dados.

Sistemas paralelos ou multi-processados são em geral sistemas que possuem dois ou mais processadores, de igual poder computacional, compartilhando alguma forma de memória: tanto memória principal (memória RAM) quanto memória secundária (discos, fitas, etc.) [26]. A literatura em Bancos de Dados Paralelos classifica as arquiteturas de hardware de máquinas paralelas a partir da configuração de acesso à memória pelos processadores. Assim, denomina de arquitetura de memória compartilhada ou "shared-memory" aquelas onde os processadores compartilham tanto memória principal, quanto memória secundária (Figura 34). Tais recursos estão disponíveis através de um barramento ou conexão de alta velocidade. Quando os processadores têm acesso exclusivo a sua memória principal e apenas a memória secundária é compartilhada, é denominada de arquitetura de disco compartilhado ou "shared-disk" (Figura 35).

Uma outra variação seria a arquitetura de memória distribuída ou "shared-nothing" (Figura 36). Aqui cada processador possui acesso exclusivo tanto à memória principal quanto à memória secundária e a comunicação é feita por troca de mensagens através de uma rede de conexão.

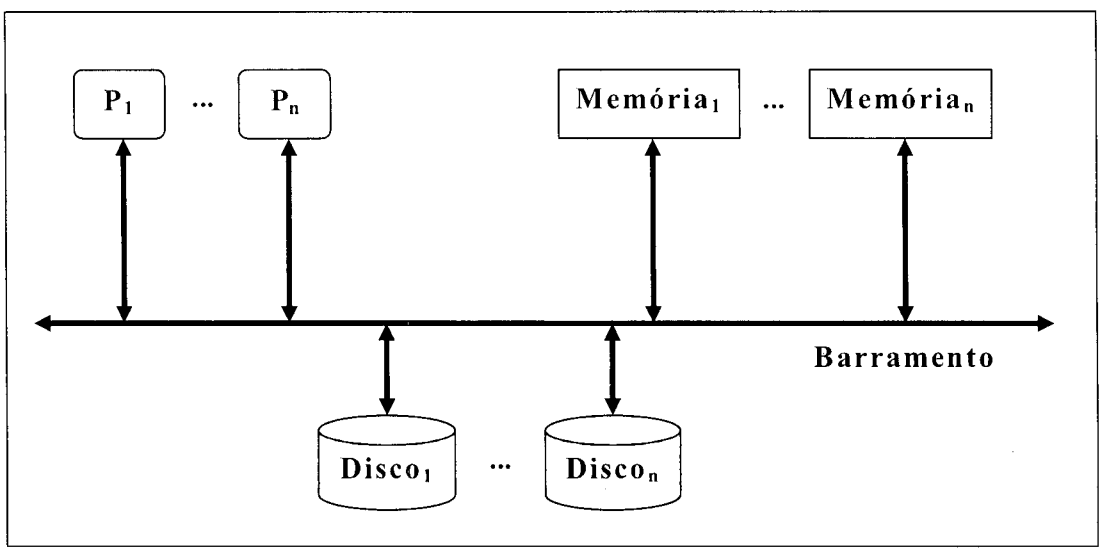

Figura 34 - Arquitetura de Memória Compartilhada

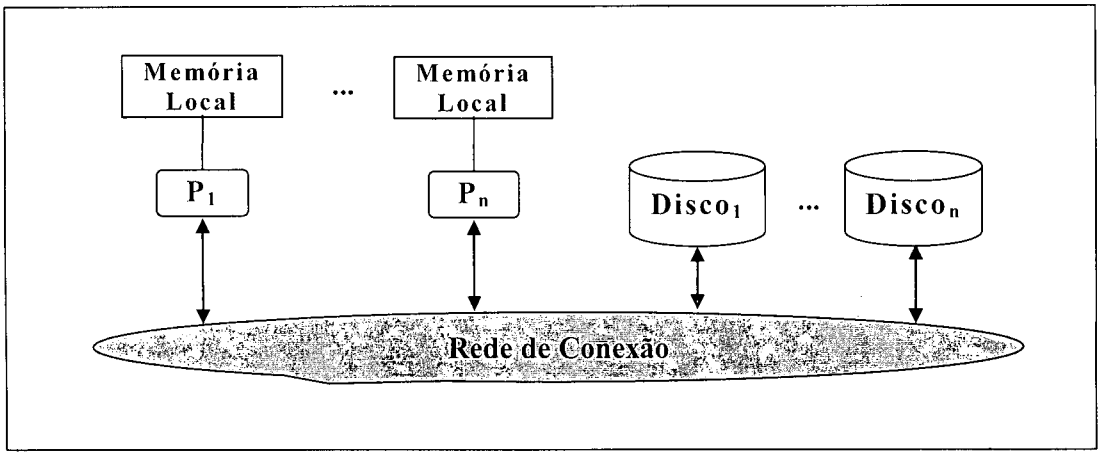

Figura 35 - Arquitetura de Disco Compartilhado

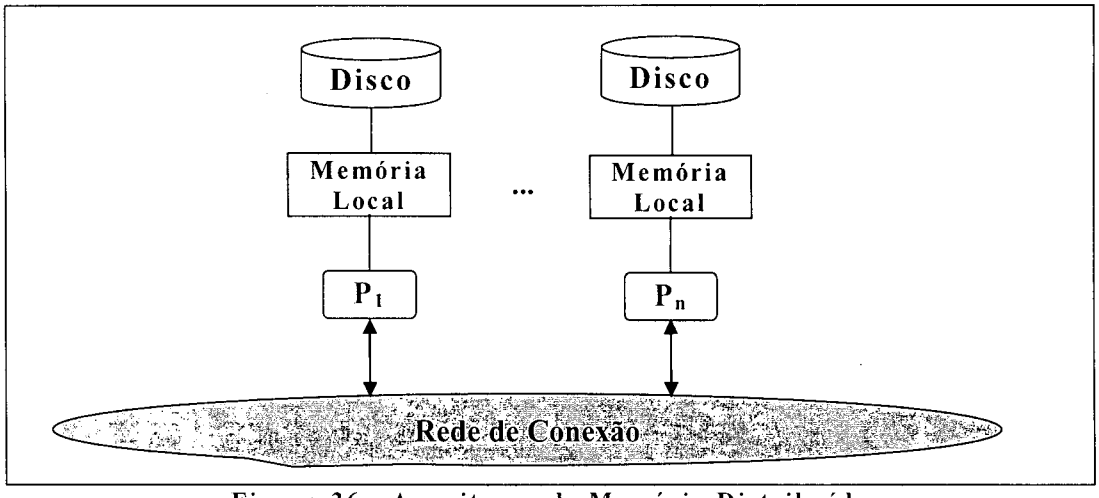

Figura 36 - Arquiteura de Memória Distribuída

Nestas arquiteturas é comum usar o termo *nó* ou "node" como representação das unidades de processamento e suas memórias. Uma característica comum às três arquiteturas é a homogeneidade dos recursos, tendo assim, processadores e memórias uma mesma configuração e capacidade. Apesar do seu uso indiscriminado o termo processamento paralelo pressupõe a homogeneidade de recursos, já o termo processamento distribuído não faz tal suposição e assim os nós envolvidos podem possuir configurações e capacidades diferenciadas [27].

Nesta dissertação e na maioria dos trabalhos de pesquisa nela avaliados, são utilizados "clusters" de computadores homogêneos como ambiente de execução e avaliação dos experimentos. A definição do que é um "cluster" pode ser bastante abrangente, por exemplo, em [28]

defini-se: "Um cluster é um tipo de sistema de processamento paralelo ou distribuído, composto por uma coleção de computadores – interconectados por uma rede local – que trabalham juntos como se fossem um único recurso computacional". Para o propósito desta dissertação um cluster se enquadra na arquitetura de memória distribuída, sendo assim composto por nós homogêneos – processadores e memória de igual capacidade – podendo também haver algum recurso compartilhado entre os nós, como unidades de disco.

Sistemas de Bancos de Dados Paralelos ou SBDP, combinam as tecnologias presentes nos SGBD – Sistemas Gerenciadores de Banco de Dados – com processamento paralelo para obter ganho de desempenho e maior disponibilidade. Um ganho significativo desta combinação é a solução geral para o problema do gargalo de Entrada e Saída ou "E/S" em dispositivos de memória secundária. Visto que o processador precisa dos dados em memória principal para poder operar, e o tempo de acesso a dispositivos de memória secundaria (discos) são muito superiores aos de acesso a memória principal, a vazão do sistema pode ficar limitada à taxa de transferência de dados do disco para a memória. Assim, uma base de dados com tamanho **D** e um único disco com vazão **T**, limita a vazão do sistema a **T**. Fragmentando **D** em **N** partes e as distribuindo a cada unidade de disco dos **N** processadores, teríamos em paralelo uma vazão de **NxT**, solucionando o problema inicial.

Apesar da possibilidade de ganho de desempenho, pesquisas na área de processamento paralelo mostraram que estes sistemas sofrem de alguns problemas que podem limitar o ganho de desempenho, entre eles temos: os custos de inicialização do sistema, a interferência entre processos por acesso simultâneo a recursos compartilhados e o balanceamento de carga. Este último tem um papel crucial em sistemas paralelos, visto que uma má distribuição da carga de trabalho ira limitar a melhora no desempenho ao tempo de resposta do nó que levar mais tempo.

Voltando à paralelização do BLAST, três estratégias principais de paralelismo vêm sendo adotadas: (i) paralelismo por hardware, (ii) por fragmentação do conjunto de seqüências de entrada (ou consulta), e (iii) por fragmentação da base de dados [29]. Destas, as duas últimas se caracterizam pelo paralelismo de dados e são as que fazem proveito das técnicas já utilizadas em SBDP. As próximas sub-seções analisam essas três estratégias.

2.3.1 Paralelismo por Hardware

As técnicas de paralelismo por hardware atuam principalmente na fase de alinhamento das seqüências. Como mencionado na seção 2.2 , a etapa de alinhamento pode ser vista como um espaço de busca. Tal representação, uma matriz **n x m**, onde **n** e **m** correspondem ao tamanho de cada uma das seqüências a serem alinhadas, facilita o emprego do paralelismo nesta fase. Em geral tais abordagens não utilizam os pacotes BLAST tradicionais [5, 20] como um módulo, pois o código fonte correspondente à fase de alinhamento tem que ser alterado de maneira que possa desfrutar das características específicas do hardware desenvolvido.

No projeto BioScan [30], um algoritmo de alinhamento semelhante à primeira versão do BLAST é implementado em um circuito VLSI "Very Large Scale Integrated" [31]. A fase de busca dos HSPs é realizada em paralelo neste circuito, e as fases de extensão e determinação da pontuação dos HSPs são realizadas por software.

Já no projeto SAMBA "Systolic Accelerator for Molecular Biological Applications" [32], um hardware específico foi desenvolvido para executar uma versão paralela do algoritmo de Smith-Waterman [18]. Assim através de determinados parâmetros seria possível executar o BLAST de forma que a fase de alinhamento fosse delegada ao SAMBA.

No projeto DeCypherBLAST [33] comercializado pela TimeLogic, um hardware específico com tecnologia FPGA "Field-Programmable Gate Array" – um tipo de microchip programável [34] – implementa a execução paralela do BLAST em uma placa PCI "DeCypherBLAST engine" que pode ser acoplada a um servidor. Através de um software específico as consultas BLAST podem ser efetuadas.

Estas abordagens apesar de eficientes, podem exigir um custo de hardware bastante elevado, bem como técnicos altamente especializados para poderem lidar com o mesmo. Outro fator importante é que o processo de manter e integrar o módulo específico de alinhamento aos pacotes BLAST já existentes também irá requerer uma equipe técnica muito especializada; alem de não poder contar com as novas versões dos pacotes públicos do BLAST, caso incompatibilidades ocorram. Por estes fatores o paralelismo por hardware pode não ser uma opção viável para muitos dos projetos de pesquisa em biologia.

2.3.2 Paralelismo por Fragmentação da Consulta.

Esse paralelismo de dados é aplicado aos dados de entrada a serem processados pelo BLAST. Esses dados de entrada compõem o que se chama de consulta, ou seja, conjunto de seqüências a serem comparadas com a base de dados indicada no BLAST. Assim, em linhas gerais, esse paralelismo é obtido ao se distribuir subconjuntos de seqüências da consulta para diferentes nós que estejam executando o BLAST. Ao final das execuções os sub-resultados são concatenados. A principal tarefa nesse paralelismo é realizar a distribuição das seqüências de modo a obter o máximo de utilização dos processadores através de um bom balanceamento de carga. Por isso, estamos chamando essa tarefa de fragmentação da consulta. Para facilitar a compreensão dos próximos tópicos, a seguinte terminologia será definida:

- E – Conjunto de nós escravos;

- E_i – i-ésimo nó de E, onde $1 \leq i \leq n$;
- M – Nó mestre;
- D – Base de dados;
- d_k – k-ésima seqüência de D, onde $1 \leq k \leq m$;
- C – Arquivo de Consultas;
- c_j – j-ésima seqüência de C, onde $1 \leq j \leq p$;
- FC_i – Fragmento disjunto de C presente no nó i, onde $C=\cup FC_i$ e **para todo** x,y,z **não existe** c_x **tal que** ($c_x \in FC_y$ **e** $c_x \in FC_z$ **e** $y \neq z$), onde $1 \leq x \leq p$, $1 \leq y,z \leq n$;
- f_h – Fragmento disjunto de D, com $1 \leq h \leq q$, onde $D=\cup f_h$ e **para todo** x,y,z **não existe** d_x **tal que** ($d_x \in f_y$ **e** $d_x \in f_z$ **e** $y \neq z$), com $1 \leq x \leq m$, $1 \leq y,z \leq q$;
- FD_i – Conjunto de fragmentos disjuntos de D presentes no nó i, com $D=\cup FD_i$;
- F_i – Conjunto disjunto de fragmentos disjuntos de D utilizados em uma determinada busca pelo nó i, onde $D=\cup F_i$ e **para todo** x,y,z **não existe** f_x **tal que** ($f_x \in F_y$ **e** $f_x \in F_z$ **e** $y \neq z$), com $1 \leq x \leq q$, $1 \leq y,z \leq n$.;
- T(X) – Função que determina o tamanho (em número de caracteres) de X, onde X pode ser uma base de dados D, um arquivo de consultas C ou uma seqüência em específico, c_j ou d_k.

O paralelismo do BLAST por fragmentação da consulta é implementado em sua maioria em "clusters" de PCs. Em geral esta estratégia está baseada, no modelo *mestre-escravo*, onde um determinado processo denominado *mestre* coordena a execução em paralelo dos outros processos, denominados *escravos*. Por questões de simplicidade será considerado que em cada nó será executado apenas um processo, apesar disto não ser um pré-requisito do modelo.

Esta estratégia segue basicamente os seguintes passos:

- Uma requisição ao BLAST é feita utilizando **C** como arquivo de consultas e **D** como base de dados;
- O nó mestre **M** – seguindo algum critério – distribui cada seqüência $\mathbf{c_j}$ de **C** entre os **n** nós escravos de **E**.
- Em cada nó $\mathbf{E_i}$ o BLAST é executado, utilizando o conjunto de seqüências $\mathbf{FC_i}$ de **C** como seqüências de consulta e **D** como base de dados. **D** é disponibilizado por replicação ou por disco compartilhado.
- Após as consultas terem sido concluídas, cada nó escravo envia ao nó mestre seus resultados, que por sua vez os consolida devolvendo a resposta à requisição inicial.

Esta última etapa é bastante simples, bastando concatenar todos os resultados parciais em um único arquivo. Na Figura 37 este esquema é apresentado, para o caso em que a base de dados é replicada.

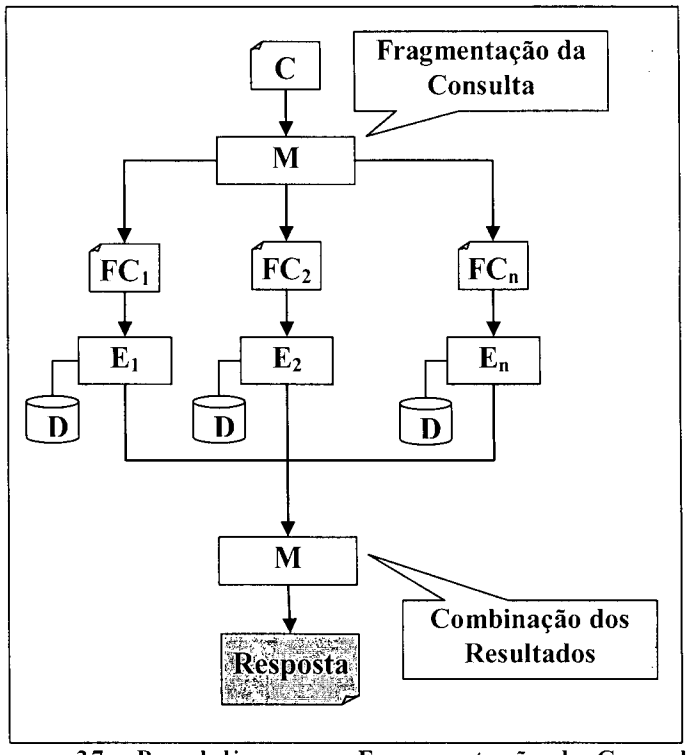

Figura 37 - Paralelismo por Fragmentação da Consulta

Uma observação importante a ser feita é de que na fragmentação do arquivo de consultas, as seqüências não são quebradas. Cada seqüência é enviada integralmente e exclusivamente a um dos **n** nós

escravos. Assim deve haver pelo menos **n** seqüências em **C** para que seja possível atribuir pelo menos uma seqüência a cada nó escravo.

Dentre os principais aspectos a serem analisados nesta estratégia, destacamos: a importância do método utilizado para distribuição das seqüências da consulta para obtenção de um bom balanceamento de carga, bem como o peso deste na inicialização do sistema; o custo da montagem dos resultados parciais oriundos dos nós escravos; e o custo de comunicação com transmissão de arquivos e/ou troca de mensagens.

Esta estratégia é implementada em [35], embora nenhuma análise de desempenho tenha sido apresentada. Neste trabalho também são feitas outras sugestões de fragmentação, incluindo a fragmentação da base de dados.

Em [36, 37], esta estratégia é implementada em um "cluster" de PCs utilizando a versão do BLAST disponibilizada pela universidade de Washington, o WU-BLAST [20]. A comunicação entre os processos é efetuada através do padrão MPI ("Message Passing Interface") [38] para troca de mensagens. A maioria dos aspectos relevantes ligados ao desempenho são analisados. Em especial, são testados diversos métodos que procuram avaliar a melhor maneira de se repartir as consultas entre os nós escravos para se obter um bom balanceamento de carga, bem como o peso de cada um na inicialização do sistema. São utilizadas seqüências geradas aleatoriamente, com tamanhos pré-determinados para compor o arquivo de consultas. A conclusão destes trabalhos é de que o melhor método é o de distribuição das seqüências sob demanda.

A grande desvantagem desta estratégia é que o tamanho da base de dados não é reduzido e assim a quantidade de acesso a disco ou E/S poderá continuar elevada em cada nó. Como visto anteriormente, este pode ser um fator crucial para a vazão do sistema, principalmente em se tratando de grandes bases de dados.

2.3.3 Paralelismo por Fragmentação da Base de Dados

Nessa estratégia de paralelismo de dados, a mesma consulta é enviada aos BLAST em execução em cada nó, para que seja comparada com diferentes porções da base de dados. O paralelismo por fragmentação da base de dados pode ser implementado tanto em "clusters" de computadores quanto em arquiteturas SMP de memória compartilhada [39].

Nas versões do NCBI-BLAST e WU-BLAST esta estratégia é implementada utilizando a arquitetura SMP, onde cada processador pesquisa uma porção da base de dados [4, 5], porém não são fornecidos maiores detalhes de como o processo se realiza. Para fazer uso desta estratégia, basta especificar na linha de comando o parâmetro '-aX' onde X especifica o número de processadores disponíveis na maquina com arquitetura SMP.

Esta estratégia, quando aplicada a arquiteturas de memória distribuída, utiliza em geral o modelo mestre-escravo, seguindo basicamente os seguintes passos:

- Uma requisição ao BLAST é feita utilizando **C** como arquivo de consultas e **D** como base de dados;
- O nó mestre **M** envia uma cópia de **C** entre os **n** nós escravos de **E**;
- Em cada nó $\mathbf{E}_i$ o BLAST é executado, utilizando **C** como seqüências de consulta; e um conjunto de fragmentos $\mathbf{F}_i$ de **D** como base de dados. Os fragmentos $\mathbf{f}_h$ de **D** são disponibilizados por replicação ou por disco compartilhado.
- Após as consultas terem sido concluídas, cada nó escravo envia ao nó mestre seus resultados, que por sua vez os consolida devolvendo a resposta à requisição inicial.

De maneira diferente da estratégia anterior, esta última etapa é um pouco mais complicada. Como cada consulta $\mathbf{c_i}$ foi realizada em **n** nós – cada um contendo um conjunto $\mathbf{F_i}$ de fragmentos de **D** – cada parte de sua resposta estará presente em cada resultado parcial gerado. Assim é necessário fazer uma consolidação dos resultados parciais, e não apenas concatená-los, para ser possível retornar à requisição inicial um resultado equivalente ao de uma execução serial do BLAST. Na Figura 38 este esquema é apresentado.

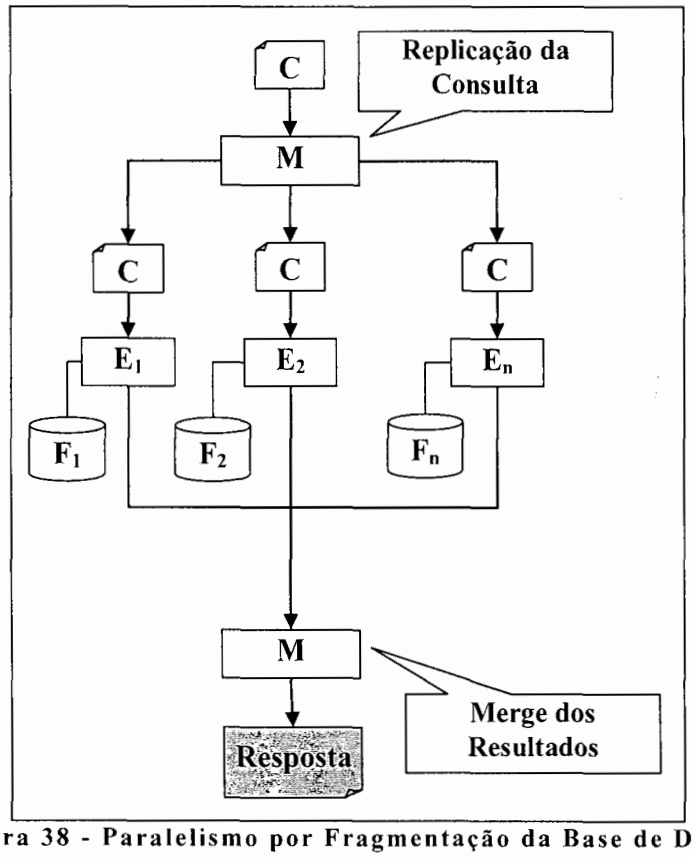

Figura 38 - Paralelismo por Fragmentação da Base de Dados

Como visto na seção 2.2 temos que o tamanho da base de dados tem influência na etapa de avaliação dos alinhamentos e assim não seria possível obter os mesmos resultados executando consultas sobre os fragmentos. Porém o BLAST possui um parâmetro que permite especificar o tamanho da base de dados a ser considerado na etapa de avaliação, o que soluciona o problema. Este parâmetro é o **z** "Effective length of the database". O seu valor por padrão é zero, o que leva o BLAST a considerar o tamanho real da base de dados.

Uma grande vantagem desta estratégia é a de poder eliminar um grande volume de entrada e saída E/S em disco nas execuções do BLAST. Como o tamanho de algumas das bases de dados atuais supera em muito, a quantidade de memória principal disponível na maioria dos computadores, o fato de cada nó estar lidando com uma fração da base de dados original, pode permitir que a mesma caiba inteiramente em memória principal, o que aceleraria em muito o tempo de busca.

Dentre os principais aspectos a serem analisados nesta estratégia, destacamos: a importância do método utilizado para fragmentação da base de dados na obtenção de um bom balanceamento de carga, bem como o peso deste na inicialização do sistema; o custo da montagem dos resultados parciais oriundos dos nós escravos; e o custo de comunicação com transmissão de arquivos e/ou troca de mensagens.

O TurboBLAST, que faz parte de um pacote comercial chamado TurboWorx [40, 41], implementa esta estratégia utilizando o NCBI-BLAST. O TurboBLAST foi projetado para ser utilizado em "clusters" de computadores heterogêneos. Ele possui um mecanismo automático de balanceamento de carga, o TurboHub, que permite um processo dinâmico de adaptação aos recursos disponíveis no "cluster". O NCBI-BLAST é utilizado praticamente sem alterações, permitindo assim a compatibilidade com futuras versões deste, bem como o uso de praticamente todas as suas funcionalidades. Contudo por ser um pacote comercial e fechado, seu uso não foi muito disseminado devido principalmente ao seu custo exorbitante e ao fato de ser difícil integrá-lo com outros aplicativos da área, conforme analisado em [42].

No "Parallel BLAST" desenvolvido pela divisão de biologia do instituto de tecnologia da Califórnia, esta estratégia é implementada utilizando o NCBI-BLAST e o padrão PVM ("Parallel Virtual Machine") [43] para troca de mensagens entre os processos. Não é fornecido explicitamente um mecanismo para balanceamento de carga e nem existe

uma integração direta com o NCBI-BLAST o que pode não garantir compatibilidade com as futuras versões deste.

Novamente em [36, 44], esta estratégia também foi implementada. Aqui foram avaliados métodos para geração dos fragmentos e o peso destes na qualidade do balanceamento de carga. O experimentos foram realizados em um "cluster" de PCs utilizando a versão do BLAST disponibilizada pela universidade de Washington, o WU-BLAST [20]. A comunicação entre os processos é efetuada através do padrão MPI ("Message Passing Interface") [38] para troca de mensagens. Chega-se a conclusão que a melhor estratégia de fragmentação é gerar fragmentos com aproximadamente o mesmo número de seqüências e caracteres. Não é levado em conta o tempo necessário para gerar estes fragmentos, bem como o custo de distribuição destes entre os nós escravos, supondo assim, que os fragmentos já estariam alocados em cada nó antes das execuções serem iniciadas. Assim, não é fornecido um mecanismo automático e dinâmico para alocação dos fragmentos em cada nó. Finalmente, o sistema não é explicitamente disponibilizado para uso e manutenção, como ocorre em projetos de software livre com código fonte aberto.

Finalmente em [4] é apresentado o mpiBLAST, uma combinação do padrão MPI com o NCBI-BLAST [5] para facilitar a implementação desta estratégia. Por ser objeto de pesquisa desta dissertação o detalhamento deste será feito no próximo capítulo.

3 Processamento Paralelo com o mpiBLAST

O mpiBLAST [4] é um projeto de software livre, com código fonte aberto, que implementa a execução em paralelo do BLAST. A estratégia do mpiBLAST consiste em fragmentar a base de dados, distribuí-la entre os nós de um "cluster", e executar a mesma consulta BLAST, simultaneamente em cada nó; cada um realizando a busca sobre uma porção única da base de dados original.

O mpiBLAST foi implementado em linguagem C e utiliza uma interface direta com a biblioteca de funções do NCBI-TOOLBOX para executar o NCBI-BLAST [5]. O modelo de comunicação entre processos é o de troca de mensagens, utilizando o padrão MPI [38] – "Message-Passing Interface". Estas características permitem que o mpiBLAST seja executado em uma grande variedade de arquiteturas.

O mpiBLAST também foi projetado para ser executado em "clusters" que possuam um sistema de gerenciamento de filas e recursos, tal como o PBS "Portable Batch System" [45]. Em tais ambientes, recursos como memória, quantidade de processadores, tempo de processamento, etc., são gerenciados pelo sistema de filas. O mpiBLAST é capaz de se adaptar aos recursos disponíveis no momento em que sua execução é iniciada redistribuindo, se necessário, os fragmentos entre os nós disponíveis.

A versão do mpiBLAST que foi utilizada nesta avaliação foi a 1.2.1, que utiliza a versão de fevereiro de 2004 do NCBI-TOOLBOX e a versão 1.2.5 do MPICH [46] para compilação.

3.1 O Algoritimo do mpiBLAST

O algoritmo do mpiBLAST consiste de duas etapas principais. Na primeira, uma base de dados **D** é fragmentada em **q** fragmentos que são colocados em um disco compartilhado. Na segunda, as consultas BLAST são executadas em cada um dos **n** nós escravos. Se um nó escravo $\mathbf{E}_i$ não possuir em disco local nenhum dos fragmentos a ser pesquisado, o mesmo copia um destes do disco compartilhado e inicia o processo de busca. A atribuição dos fragmentos a cada nó é determinada por um algoritmo que visa minimizar o número de cópias de fragmentos do disco compartilhado para os nós, durante cada busca.

3.1.1 Geração dos Fragmentos da Base de Dados

A geração dos fragmentos da base de dados é realizada por um aplicativo chamado *mpiformatdb*, onde se especifica o número ou tamanho desejado para os mesmos. O mpiformatdb é basicamente uma interface, que gera os parâmetros necessários para o formatdb produzir os fragmentos. Como descrito na seção 2.2 o formatdb é um aplicativo fornecido pelo NCBI TOOLBOX, que transforma uma base de dados no formato FASTA, em uma base de dados BLAST. Caso seja especificado o número de fragmentos, o mpiformatdb lê a base de dados, estima o tamanho destes, e invoca o formatdb passando o tamanho obtido como parâmetro, que por sua vez gera os fragmentos. Um processo equivalente acontece, caso seja especificado o tamanho desejado para os fragmentos, a única diferença é que não ocorre a fase em que o mpiformatdb estima o tamanho dos mesmos (Figura 39).

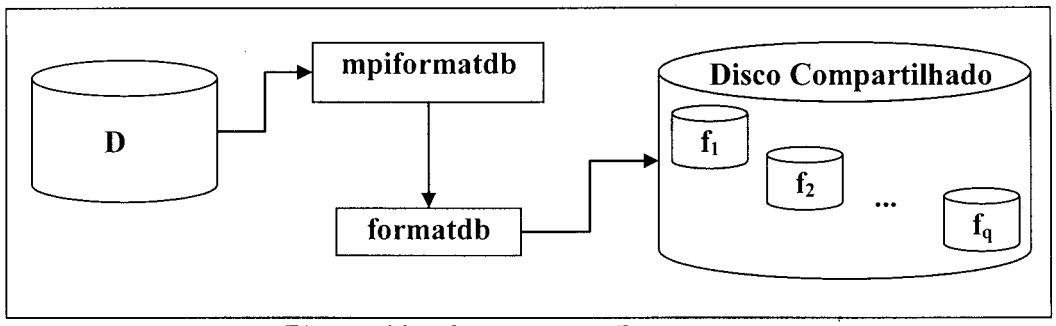

Figura 39 - Geração dos Fragmentos

3.1.2 Execução do mpiBLAST

A execução do mpiBLAST se inicia ao fazer a chamada ao ambiente mpi – *mpirun* – invocando a execução do mesmo. Nesta chamada é especificado o número de processos a serem criados e o número de nós disponíveis. A situação ideal é ter um número de processos igual ao número de nós, acrescido de mais uma unidade e assim é executado apenas um processo em cada um, com exceção do primeiro nó que executará também o processo mestre. Como feito anteriormente será suposto aqui que há somente um processo sendo executado em cada nó.

Após o inicio da execução do mpiBLAST, cada um dos **n** nós escravos informa ao nó mestre quais fragmentos da base de dados a ser consultada já estão armazenados em disco local, o conjunto $\mathbf{FD}_i$. Esta informação fica armazenada em um arquivo, cujo nome segue o seguinte padrão: 'nome_da_base_de_dados.mbf'. Em seguida o nó mestre **M** envia o arquivo de consultas **C** para os **n** nós escravos. Quando esta etapa se conclui, cada nó escravo reporta ao nó mestre que está ocioso; este ao receber a mensagem de ociosidade, atribui um fragmento da base de dados para o nó escravo pesquisar. O nó escravo então verifica se possui o fragmento em disco local, caso não possua, realiza uma cópia do fragmento do disco compartilhado para o disco local e em seguida efetua a busca. Encerrada a busca, o nó escravo envia uma mensagem ao nó mestre, informando novamente que está ocioso. Este procedimento se repete até que todos os fragmentos da base de dados tenham sido

pesquisados. Na Figura 40 é apresentado este esquema de execução mpiBLAST.

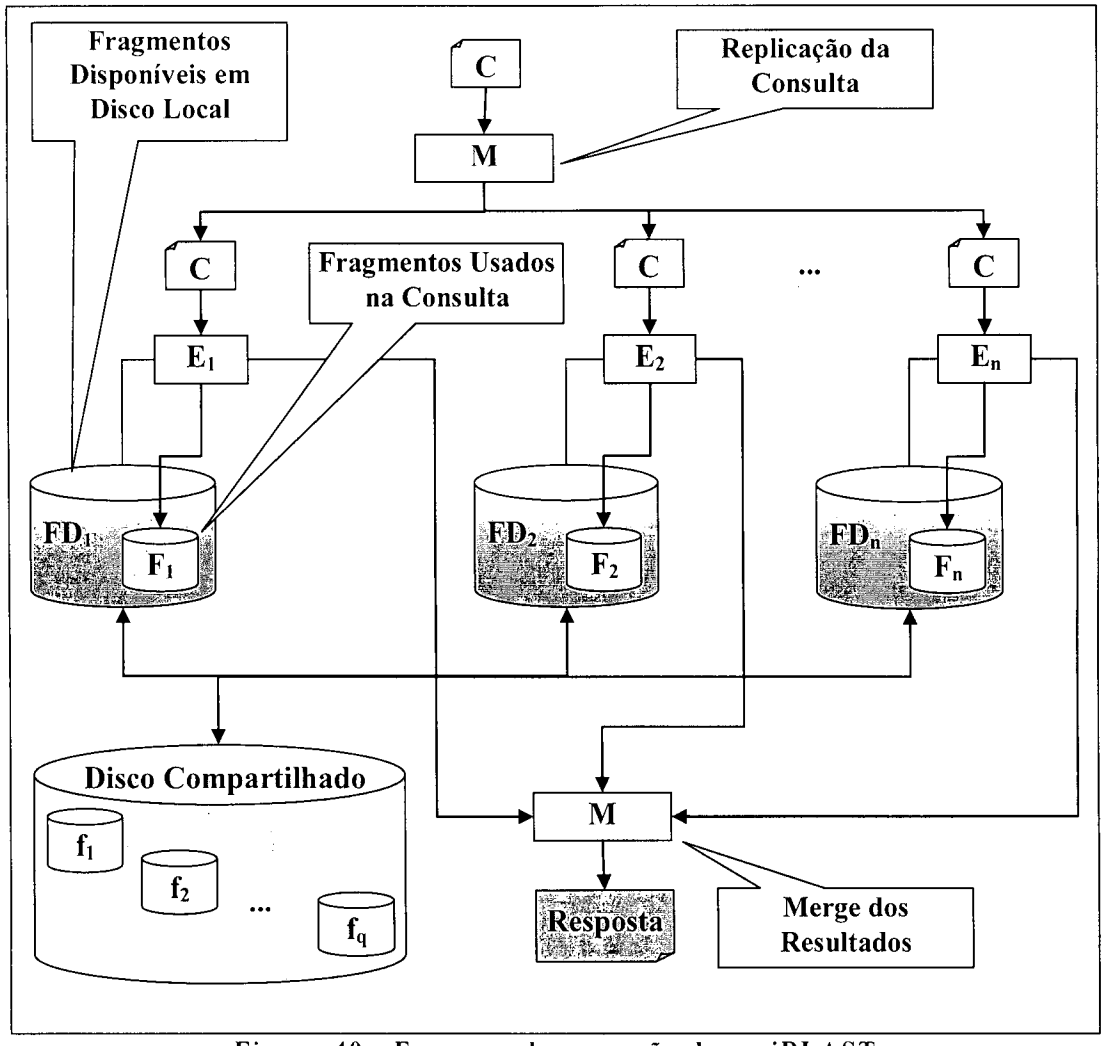

Figura 40 - Esquema de execução do mpiBLAST

O nó mestre **M** utiliza um algoritmo guloso para determinar quais fragmentos devem ser atribuídos a cada nó escravo. Primeiramente é verificado se o nó ocioso possui algum fragmento ainda não pesquisado, que nenhum outro nó o tenha; caso possua, ele irá pesquisar este fragmento, caso contrário é atribuído aquele fragmento que existir em menor quantidade em outros nós. O Conjunto de fragmentos que estão sendo copiados é monitorado pelo nó mestre de forma a evitar que cópias duplicadas sejam efetuadas em diferentes nós escravos em uma mesma rodada. Na Figura 41 é apresentado este algoritmo como descrito em [4].

Algorithm 1 mpiBLAST master:

Let results be the current set of BLAST results
Let **F** = $\{f_1, f_2, ...\}$ be the set of database fragments
Let **Unsearched** $\subseteq$ **F** be the set of unsearched database fragments
Let **Unassigned** $\subseteq$ **F** be the set of unassigned database fragments
Let **W** = $\{w_1, w_2, ...\}$ be the set of participating workers
Let $\mathbf{D}_i \subseteq$ **W** be the set of workers that have fragment f_i on local storage
Let **Distributed** = $\{\mathbf{D}_1, \mathbf{D}_2, ...\}$ be the set of **D** for each fragment

Require: |**W**| $\neq 0$
Ensure: |**Unsearched**| = 0
Unsearched ← **F**
Unassigned ← **F**
results ← 0
Broadcast queries to workers
while |**Unsearched**| $\neq 0$ **do**
receive a *message* from a worker w_j
if *message* is a state request **then**
if |**Unassigned**| = 0 **then**
Send worker w_j the state *SEARCH_COMPLETE*
else
Send worker w_j the state *SEARCH_FRAGMENT*
end if
else if *message* is a fragment request **then**
Find f_i such that min(|$\mathbf{D}_i$|, $\mathbf{D}_i \in$ **Distributed**) **and** $f_i \in$ **Unassigned**
if |$\mathbf{D}_i$| = 0 **then**
Add w_j to $\mathbf{D}_i$
end if
Remove f_i from **Unassigned**
Send fragment assignment f_i to worker w_j
else if *message* is a set of *search results* for fragment f_i **then**
Merge *message* with *results*
Remove f_i from **Unsearched**
end if
end while
Print *results*

Algorithm 2 mpiBLAST worker:

queries ← Receive the queries from the master
currentState ← Receive the state from the master
while *currentState* $\neq$ *SEARCH_COMPLETE* **do**
currentFragment ← Receive a fragment assignment from the master
if *currentFragment* is not on local storage **then**
Copy *currentFragment* to local storage
end if
results ● *BLAST(queries, currentFragment)*
Send *results* to master
currentState ← Receive the state from the master
end while

Figura 41 - Algoritmo do mpiBLAST

3.2 Análise do mpiBLAST e Propostas

Neste capítulo é feita uma análise do mpiBLAST relacionada a alguns dos aspectos mencionados como relevantes no capítulo anterior, relembrando:

- A importância do método utilizado para fragmentação da base de dados na obtenção de um bom balanceamento de carga, bem como o peso deste na inicialização do sistema;
- O custo da montagem dos resultados parciais oriundos dos nós escravos;
- Detecção de possíveis pontos de estrangulamento na execução do algoritmo e no acesso a recursos compartilhados.

Algumas outras observações que não estejam relacionadas ao ganho de desempenho mais que possam trazer benefícios ao sistema também são feitas como, por exemplo, alternativas para tolerância à falhas, interação com workflows, etc.

3.2.1 Geração dos Fragmentos

O processo de geração dos fragmentos no mpiBLAST, constitui uma etapa preliminar a execução do algoritmo. Ao basear sua estratégia apenas na fragmentação da base de dados, um bom balanceamento de carga somente será alcançado com uma boa escolha do número de fragmentos, bem como uma boa distribuição das seqüências da base de dados dentro destes fragmentos. Assim a etapa de geração dos fragmentos passa a exercer um papel fundamental na obtenção de um bom desempenho do sistema.

Conforme já visto, no mpiBLAST, o aplicativo responsável por esta tarefa é o *mpiformatdb*. Ele oferece duas opções principais para geração dos fragmentos, que são a escolha do tamanho esperado para os

fragmentos ou a escolha do número de fragmentos esperado. A primeira opção irá gerar o número de fragmentos necessário para que estes possuam aproximadamente o tamanho especificado. Já a segunda opção irá gerar fragmentos de aproximadamente o mesmo tamanho. Em ambos os casos ter o mesmo tamanho implica em ter mais ou menos o mesmo número de caracteres.

Para obter o máximo de proveito do paralelismo oferecido pelo mpiBLAST, o mesmo considera como situação ideal para sua execução ter:

- Número de Fragmentos (**q**) = Número de nós (**n**);
- Fragmentos de aproximadamente o mesmo tamanho;
- Número de processos = **n** + **1**.

Utilizando a primeira opção do mpiformatdb, podemos garantir os dois primeiros itens desta configuração, porém o fato de gerar fragmentos de aproximadamente o mesmo tamanho, pode não garantir um bom balanceamento de carga. A principal razão para isto é que a comparação de uma dada seqüência **a**, com outras duas seqüências de mesmo tamanho **b**, e **c**, pode levar à tempos de processamento diferentes. Tal fato se deve ao grau de similaridade entre as seqüências. Se a seqüência **a** possuir um grau de similaridade maior com a seqüência **b** do que com a seqüência **c**, a fase dois do BLAST – A Extensão dos Acertos – irá demorar mais entre as seqüências **a** e **b** do que entre as seqüências **a** e **c**.

Análises feitas através do "GNU Profiler" ou *gprof* [47] – o gprof gera um perfil de execução da aplicação, permitindo que seja observado quais rotinas de um programa são invocadas na execução deste, bem como o tempo gasto em cada uma delas – demonstraram que cerca de 75 a 80% do tempo de execução do NCBI-BLAST são gastos na rotina *BlastExtendWordSearch* (Figura 42) que pertence a rotina *BlastWordFinder_mh_contig* presente no arquivo *blast.c* da biblioteca de funções do NCBI-BLAST [48]. Esta rotina é responsável basicamente

pela execução das fases um e dois do BLAST. Assim se houver uma maneira de se evitar estas fases seria possível obter um ganho de tempo e processamento.

Flat profile:
Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self Ks/call	total Ks/call	name
76.82	1226.53	1226.53	285541639	0.00	0.00	BlastExtendWordSearch
11.41	1408.64	182.11	285541638	0.00	0.00	BlastTranslateUnambiguousSequence
4.30	1477.34	68.70	47590272	0.00	0.00	BLASTPerformFinalSearch
4.06	1542.16	64.82	310887726	0.00	0.00	BlastWordExtend_prelim

Figura 42 - Perfil de Execução do BLAST Gerado pelo gprof

Apesar da duração do passo dois do BLAST estar relacionado ao grau de similaridade entre as seqüências, esta não pode ser estimada a priori. Conforme descrito na seção 2.2 , podemos verificar que na fase de escolha das sementes, seqüências maiores terão uma probabilidade maior de obter um número maior de acertos. Assim por conseqüência do maior número de acertos encontrados na fase um do BLAST, há também um aumento na probabilidade de haver a segunda fase do algoritmo que seria a extensão destes. Como a maior parte do tempo de execução do BLAST é gasto nestas duas fases, uma distribuição mais uniforme das seqüências grandes, médias e pequenas entre os fragmentos seria uma boa heurística para a geração dos fragmentos na tentativa de se obter um melhor balanceamento de carga.

Baseado nesta heurística, podemos então questionar a eficácia do *mpiformatdb* em gerar fragmentos de aproximadamente o mesmo tamanho. Conforme visto no parágrafo anterior, um dado fragmento $\mathbf{f_x}$ poderá possuir **y** seqüências 'grandes' e um outro fragmento $\mathbf{f_z}$, **w** seqüências 'pequenas', onde **w >> y**, embora o tamanho $\mathbf{T(f_x)}$ em número de caracteres do fragmento $\mathbf{f_x}$ seja aproximadamente igual ao do fragmento $\mathbf{f_z}$. Assim consultas ao fragmento $\mathbf{f_x}$ tenderiam a ser bem mais demoradas do que quando submetidas ao fragmento $\mathbf{f_z}$, levando a um desbalanceamento de carga.

O algoritmo aqui proposto tenta levar proveito desta informação e assim, procura gerar fragmentos com aproximadamente o mesmo tamanho

e com aproximadamente o mesmo número de seqüências grandes, médias e pequenas. A estratégia básica é pré-ordenar as seqüências por ordem decrescente de tamanho e em seguida distribuí-las entre os fragmentos (Figura 43).

Seja:

D – Base de dados ou arquivo de seqüências biomoleculares;

$\mathbf{d_k}$ – k-ésima seqüência de D, $1 \leq k \leq m$;

T(X) – Função que determina o numero de caracteres presente em uma base de dados X, ou em uma determinada seqüência do banco de dados X;

$\mathbf{f_h}$ – h-ésimo fragmento de D, $1 \leq h \leq q$;

LD – Lista das m seqüências de D ordenada de forma crescente por $T(d_k)$. Cada elemento desta lista possui dois campos: **identificador**, referente ao índice k de cada seqüência da base de dados D e o campo **tamanho**, referente ao numero de caracteres deste fragmento;

LF – Lista dos m fragmentos de D ordenada de forma crescente por $T(f_h)$ e pelo numero de seqüências presentes em f_h. Cada elemento desta lista possui três campos: **identificador**, referente ao identificador h de cada fragmento; o campo **tamanho**, referente ao numero de caracteres presente em cada fragmento; e o campo **num_seq**, referente ao numero de seqüências presente neste fragmento;

OrdenaLD – Procedimento que ordena a lista LD da forma especificada anteriormente;

OrdenaLF – Procedimento que ordena a lista LF da forma especificada anteriormente;

Adiciona(X, Y) – Procedimento que adiciona ao fragmento X a seqüência Y;

```
:Inicio
        j = 1;
        i = 1;
        enquanto i ≤ m faça
                LF[i].indentificador ← i;
                LF[i].tamanho ← 0;
                LF[i].num_seq ← 0;
                i = i +1;
        OrdenaLD;
        OrdenaLF;
        enquanto j ≤ n faça
                LF[1].tamanho ← LF[1].tamanho + T(D_LD[j].identificador);
                LF[1].num_seq ← LF[1].num_seq +1;
                Adiciona(F_LF[1].identificador, D_LD[j].identificador);
                OrdenaLF;
                j = j +1;
:Fim
```

Figura 43 - Algoritmo de Geração dos Fragmentos

A ordem decrescente foi escolhida, pois é mais fácil ajustar o tamanho dos fragmentos com seqüências cada vez menores do que com o contrário.

Uma análise do custo desta ordenação em relação ao tempo de processamento deve ser feita para verificar se o ganho de desempenho compensa. Neste caso deve-se também levar em conta a freqüência com que à base de dados é atualizada e o volume de consultas realizado sobre esta.

Apesar de não ser um empecilho ao seu funcionamento, visto que a distribuição dos fragmentos é dinâmica, o mpiBLAST foi projetado para ser executado em um ambiente *estável*. Aqui o termo estável se refere a um ambiente em que o número de nós disponíveis é aproximadamente constante e a geração dos fragmentos pode ser efetuada poucas vezes e de forma a gerar um número de fragmentos **q** equivalente ao número de nós **n**, atingindo assim a situação ideal para o seu funcionamento **n=q**. Visto que para bases de dados grandes a fase de construção dos fragmentos irá requerer um razoável tempo de processamento, em ambientes não estáveis, a geração dos fragmentos poderá comprometer o desempenho total do sistema. Nas situações em que: **($n \neq p$)** e **($n \div q \neq 0$)** a ociosidade de alguns nós poderá degradar o desempenho de execução do mpiBLAST. Duas abordagens poderiam ser adotadas para contornar este problema. A primeira e mais simples, pois não exigiria modificações no atual algoritmo do mpiBLAST, seria gerar um número maior de fragmentos pequenos. A segunda e mais complexa, pois exigiria modificações no atual algoritmo, seria fragmentar as seqüências de consulta e distribuí-las sob demanda.

Na primeira proposta, a de gerar um número maior de fragmentos pequenos, teríamos que nos casos em que **($n \div q \neq 0$)** mesmo havendo ociosidade em alguns processadores, este tempo seria bem menor. Uma questão importante é que um número maior de execuções do BLAST

também será realizado, visto que o número de fragmentos cresceu, o que pode aumentar o tempo de processamento [4].

3.2.2 Consolidação dos Resultados

Na execução do mpiBLAST, uma consulta é submetida aos **q** fragmentos gerados a partir da base de dados original **D**. Assim, em cada fragmento é executado um BLAST utilizando uma mesma seqüência como consulta, após todos os fragmentos terem sido pesquisados os resultados individuais de cada execução são consolidados no nó mestre **M**. Como já visto, tal consolidação dos resultados é necessária para que um resultado equivalente ao da execução de um BLAST serial seja gerado. Dependendo do custo de se fazer esta montagem em relação ao tempo total de processamento, esta etapa poderia ser considerada um gargalo no sistema, visto que todo processamento se concentra em **M**, além de haver um congestionamento no envio de mensagens dos resultados dos **n** nós escravos para o nó mestre **M** (Figura 44).

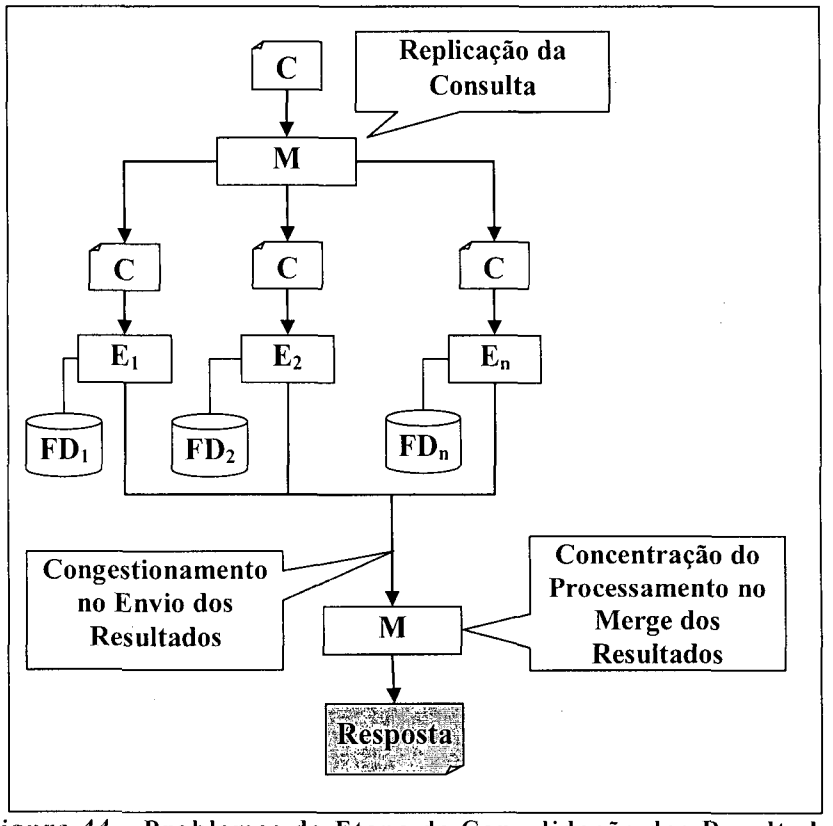

Figura 44 - Problemas da Etapa de Consolidação dos Resultados

Uma alternativa interessante seria dividir esta tarefa com os outros nós escravos, que ficam ociosos durante esta etapa. Para evitar alterações mais profundas no algoritmo atual, poderia ser usada uma estratégia mista, com fragmentação da consulta e da base de dados. Para isso os **n** nós disponíveis seriam divididos em **w** partes, e **w** fragmentos do arquivo de consultas **C** seriam gerados. Sendo **u** = **n÷w**, seriam gerados **u** fragmentos de **D**. Com esta configuração, **w** execuções em paralelo do mpiBLAST seriam realizadas, cada uma delas utilizando um dos fragmentos de **C** como consulta e os **u** fragmentos de **D** como base de dados (Figura 45). Ao final bastaria concatenar os resultados, o que é trivial.

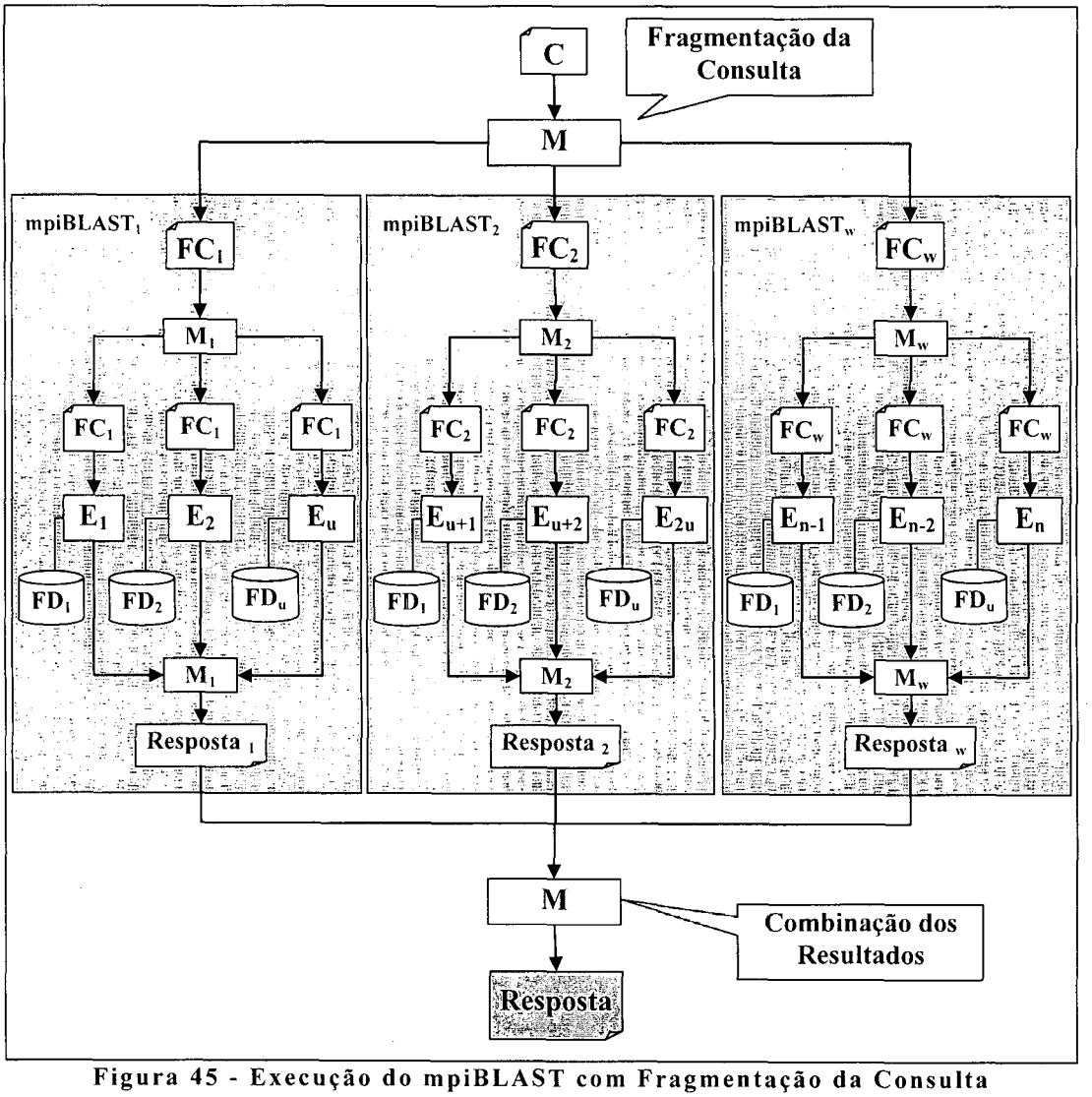

Figura 45 - Execução do mpiBLAST com Fragmentação da Consulta

Nesta estratégia uma questão bastante relevante é o aumento do tamanho dos fragmentos. Dado que são gerados **u** fragmentos ao invés de **n** e **u** = **n÷w**, então o tamanho dos fragmentos nesta estratégia será **w** vezes maior do que na estratégia anterior. Assim deve-se avaliar um bom valor para **w** de maneira que as buscas possam ser realizadas em memória principal.

3.2.3 Outras Considerações

Conforme já mencionado, é possível submeter diversas consultas de uma só vez ao BLAST, para isso basta juntá-las em um único arquivo de consulta **C** antes de submetê-las. Apesar disso, as **q** seqüências de **C** serão avaliadas uma de cada vez pelo BLAST. Contudo, devido aos custos de inicialização do sistema, submeter as **q** seqüências de **C**, uma de cada vez, fazendo assim **q** chamadas BLAST, é uma procedimento mais lento que o anterior. Assim, caso o objetivo seja reduzir o tempo total de execução – tempo necessário para ultima seqüência de **C** ser processada – o primeiro procedimento é mais apropriado. Porém se o objetivo for reduzir o tempo de resposta – tempo necessário para que os primeiros resultados sejam gerados – o segundo procedimento será mais interessante, pois o tempo necessário para processar cada uma das seqüências é relativamente curto.

Priorizar a redução do tempo de resposta pode ser uma boa estratégia quando é possível realizar algum trabalho sobre os resultados na medida em que estes são gerados. O uso do BLAST em workflows científicos seria um exemplo desta situação. Em [49] este caso é caracterizado como *Paralelismo Intra-Workflow* (Figura 46). Neste trabalho é apresentado o "MholLine" (Figura 47) um workflow biológico que combina um conjunto de programas de bioinformática – entre eles o BLAST – para obter modelos em 3D de proteínas.

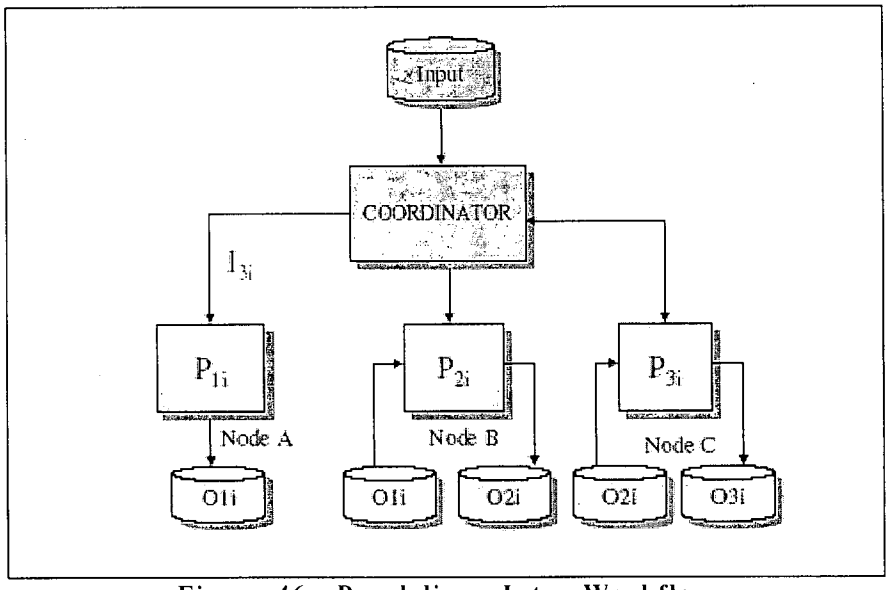

Figura 46 - Paralelismo Intra-Workflow

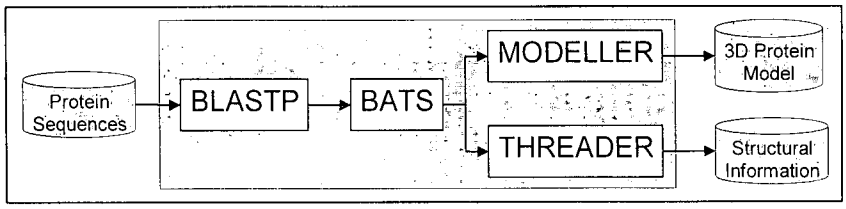

Figura 47 - MholLine

A viabilização desta estratégia seria trivial, bastando fragmentar o arquivo de consultas com o número de seqüências desejados em cada fragmento e submetê-los um a um.

Ainda nesta questão de fragmentar o arquivo de consultas, uma segunda observação poderia ser feita no que diz respeito à amenização dos efeitos de uma possível falha durante a execução do programa. Como o mpiBLAST não possui um mecanismo de tolerância a falhas e dado que o tempo de uma submissão pode vir a ser bastante longo, a submissão das consultas por partes seria uma alternativa simples para evitar que os resultados das consultas já processadas fossem perdidos na eventualidade de uma falha.

4 Implementação e Análise dos Resultados

4.1 Ambiente de execução dos exeperimentos

Os experimentos realizados nesta dissertação foram executados no "cluster" Itautec do NACAD (Núcleo de Atendimento em Computação de Alto Desempenho) da COPPE/UFRJ. O "cluster" é composto por 16 nós dual processados, Intel Pentium III de 1GHz. Cada nó possui 512MB de memória RAM e 18GB de espaço em disco, além de 256KB de memória cache por CPU. Existe também um nó a mais, denominado *estação de administração* ou *adm*, que possui a mesma configuração dos outros nós, porém, com 160GB de espaço em disco. Este disco da estação adm é compartilhado com os outros nós do "cluster", via NFS "network file system". A rede de comunicação entre os nós utiliza tecnologia Fast-Ethernet (100 Mbits/s) e o sistema operacional é Linux distribuição RedHat 7.3.

As submissões são feitas ao "cluster" através da estação adm, estas obedecem a um sistema de filas, o PBS ("Portable Batch System"). Este sistema controla os recursos disponíveis, como memória, número de nós, número de processadores, tempo de processamento, etc., e os recursos solicitados por cada execução submetida. Assim um processo fica aguardando na fila até que os recursos solicitados estejam disponíveis e, quando em execução, tem a garantia de que não sofrerá interferência de outros processos, facilitando assim a coleta de medidas.

As implementações necessárias foram realizadas em linguagem C++, utilizando o padrão STL "Standard Template Library" [50] e o compilador GCC "GNU Compiler Collection" versão 2.96 [51].

A versão do mpiBlast utilizada nos testes foi a 1.2.1 que utiliza a versão de fevereiro de 2004 do NCBI-TOOLBOX e a versão 1.2.5 do MPICH para compilação.

As seqüências de consulta utilizadas nos experimentos foram obtidas de dados reais de pesquisa da Fundação Oswaldo Cruz, no Rio de Janeiro. Esta escolha foi feita de modo que se evitasse a geração de dados aleatórios que pudessem não refletir uma situação real. Na Tabela 5 é exibido um sumário destas seqüências obtidas.

Tabela 5 - Dados das Sequencias de Consulta

Número de Sequencias	1.154
Número de pares de base (bp)	541.089
Tamanho da Maior Sequencia em bp	2.075
Tamanho da Menor Sequencia em bp	151

A escolha da base de dados foi feita basicamente levando em consideração o seu tamanho. Por ser uma das maiores, a base de dados escolhida foi a do *nt*, que contém seqüências de nucleotídeos oriundas das bases do GenBank, EMBL e DDBJ. Esta base foi obtida do NCBI em Dezembro de 2004 e o arquivo no formato FASTA possuía cerca de 12.9GB* de tamanho. Na Tabela 6 é exibido um sumário desta base de dados.

Tabela 6 - Dados da Base nt

Número de Sequencias	2.910.708
Número de pares de base (bp)	13.358.659.363
Tamanho da Maior Sequencia em bp	43.261.740
Tamanho da Menor Sequencia em bp	6

Conforme mencionado na seção 2.2.1 o BLAST possuí cinco *sabores* ou tipos de busca. Dentre eles o tBLASTx é o mais custoso e demorado, tendo uma grande demanda por capacidade de processamento e por memória. A principal razão disto é que tanto as seqüências da base de dados quanto às seqüências de consulta são traduzidas, de nucleotídeos para aminoácidos, antes da busca se realizar. Relembrando que para compor cada aminoácido são necessários três nucleotídeos, temos que fazendo a leitura de três em três pares de base, é possível

* ftp.ncbi.nih.gov/blast/db/FASTA/nt.gz

gerar três diferentes traduções, uma começando pela primeira base da seqüência, a seguinte começando pela segunda e a última começando pela terceira. Estas três possíveis traduções são denominadas "frames" ou *quadros de leitura* como cada seqüência também possui dois sentidos de leitura, 5' → 3'e 3' → 5', temos então seis possíveis quadros, três no primeiro sentido, e três no segundo. Na Figura 48 é apresentado um exemplo deste processo, os quadros são identificados com um sinal de mais ou menos, seguido de um número – um, dois ou três – que indica a posição de inicio da leitura [52]. Os quadros negativos são obtidos da cadeia no sentido 3' → 5' mais o complemento de suas bases. Para cada quadro a linha superior corresponde a cadeia de nucleotídeos e a linha inferior a sua tradução em aminoácidos.

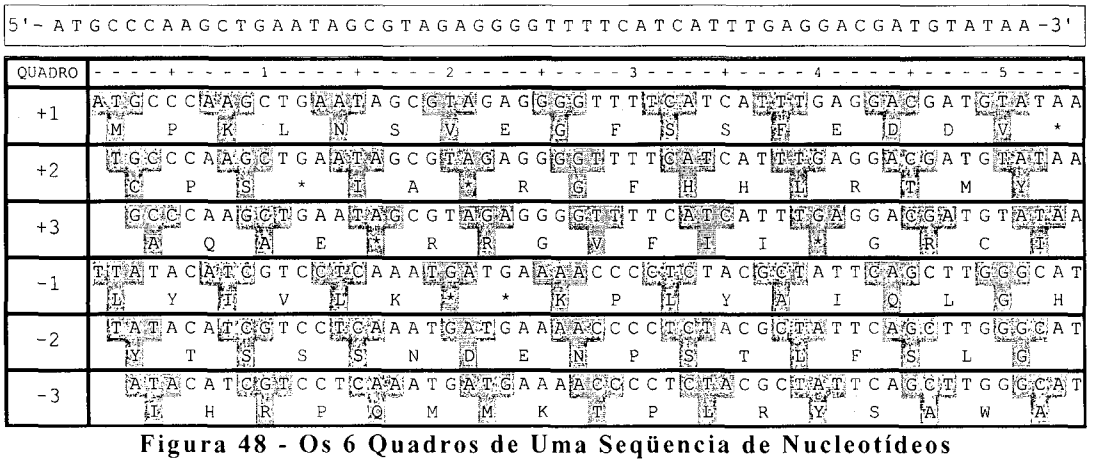

Figura 48 - Os 6 Quadros de Uma Seqüencia de Nucleotídeos

Dado que este processo ocorre tanto para a consulta quanto para a base de dados, por combinação, é como se o tBLASTx efetuasse trinta e seis buscas do tipo BLASTp – tipo de busca BLAST quando consulta e base de dados são de aminoácidos – justificando assim o seu elevado custo de processamento.

Uma execução serial do BLAST foi realizada e seu tempo de execução foi medido com o objetivo de compará-lo com os tempos obtidos nas estratégias paralelas. Esta primeira execução levou 706:11:44 hh:mm:ss ou cerca de 30 dias. Por ocasião de restrições relativas ao tempo de uso disponível no "cluster" versus o número de

rodadas necessário para avaliar os experimentos, decidiu-se usar apenas uma parte da base de dados original do *nt*. A nova base passou a conter aproximadamente os 2.0GB iniciais do arquivo original e o tempo de execução serial foi reduzido para 101:35:14 hh:mm:ss ou aproximadamente 4 dias e meio. Na Tabela 7 é exibido um sumário desta base de dados

Tabela 7 - Dados da Base nt Reduzida

Número de Sequencias	574.855
Número de pares de base (bp)	2.055.228.948
Tamanho da Maior Sequencia em bp	6.021.225
Tamanho da Menor Sequencia em bp	6

4.2 Estratégia para geração dos Fragmentos

4.2.1 Método de Avaliação

Para avaliar a eficácia da estratégia atual de geração de fragmentos empregada pelo mpiBLAST, serão adotadas três versões da base de dados *nt*: 'nt.nat', 'nt.ord' e 'nt.pro'.

A primeira delas, 'nt.nat', é idêntica a base original, aqui a extensão 'nat' faz menção a base possuir as seqüências em sua ordem natural ou original, ou seja, na ordem em que estas foram adicionadas ao arquivo pelo NCBI. A segunda, 'nt.ord', possui suas seqüências em ordem decrescente de tamanho, e a extensão 'ord' faz menção a isto. Nestas duas versões, os fragmentos serão gerados pelo próprio mpiBLAST, através do *mpiformatdb*. O objetivo com isto é verificar como o aplicativo irá distribuir as seqüências entre os fragmentos. Espera-se que a distribuição do número de seqüências da base 'nt.ord' seja bastante desiguais.

Já na terceira e última versão, a 'nt.pro', os fragmentos serão gerados segundo a ordem do algoritmo proposto na secção 3.2.1 e em

seguida serão formatados pelo aplicativo *formatdb* para que os fragmentos fiquem no formato de uma *base de dados BLAST*.

Os testes de cada versão da base consistiram de quatro rodadas, com 2, 4, 8 e 16 nós, tendo em cada uma delas um número de fragmentos equivalente ao número de nós.

4.2.2 Detalhes de Implementação

Nesta seção são feitas algumas observações a respeito da implementação do algoritmo responsável pela geração dos fragmentos da versão 'nt.pro' da base de dados. Como descrito na seção relativa ao ambiente de implementação, o algoritmo foi desenvolvido na linguagem C++. A biblioteca STL foi utilizada para manipulação das estruturas de dados, como por exemplo, árvores, filas, etc.

Devido ao tamanho das bases de dados, não era possível nem aconselhável fazer a ordenação das seqüências em memória. Assim decidiu-se construir um índice, onde cada entrada ou registro possuía os seguintes campos: *tamanho*, campo correspondente ao tamanho da seqüência; *início*, campo correspondente ao valor do byte onde a seqüência se inicia no arquivo; e *fim*, campo correspondente ao valor do byte onde a seqüência termina no arquivo. Assim, em um processo de varredura da base de dados, as seqüências eram lidas e as entradas do índice eram determinadas. As inserções no índice eram feitas de forma a mantê-lo sempre ordenado, no caso em ordem decrescente de tamanho. A estrutura de dados utilizada para representar o índice, foi uma árvore binária, fornecida pela estrutura "MAP" da STL [50].

Uma vez construído o índice, suas entradas eram distribuídas entre os fragmentos, segundo o critério do algoritmo proposto. Aqui cada fragmento foi representado como uma classe que possuía os campos *tamanho*, referente ao número de caracteres e o campo *num_seq*, referente ao número de seqüências. Além disso, cada fragmento também

possui uma estrutura equivalente ao índice principal, porém neles as entradas são inseridas e mantidas ordenadas pela posição do byte de início no arquivo. Tal estratégia foi necessária para minimizar os acessos aleatórios a base de dados original no momento em que os fragmentos são gravados em disco, com isso também é possível ler o arquivo em uma única direção, o que pode ser útil em uma futura política de cache.

Uma implementação semelhante a esta também foi utilizada para gerar a versão 'nt.ord' da base, onde não são gerados fragmentos e o arquivo – e não o índice – é realmente reordenado. Esta implementação foi necessária somente para a fase de testes.

Um problema que surgiu durante a fase de implementação, foi a capacidade de endereçamento às posições dos bytes no arquivo. Por usarem o tipo inteiro de 32 bits com sinal, as funções tradicionais da biblioteca "fstream e iostream" de manipulação de arquivos do C++, somente conseguiam endereçar posições até 2^{31}-1, o que permite lidar com arquivos de até 2GB, embora o sistema operacional suportasse arquivos maiores. Na verdade, não ocorria erro de leitura, mas as posições dos bytes no arquivo eram perdidas na medida em que seu valor superasse a precisão do tipo inteiro e assim o índice perdia sua finalidade. Depois de detectado, o problema foi sanado utilizando outras rotinas, mais básicas, da biblioteca "stdio.h". Estas rotinas utilizam um tipo especial, que emula inteiros de 64 bits e assim a capacidade de endereçamento é resolvida. O único inconveniente que ainda permaneceu foi que mesmo usando os compiladores GCC, o mesmo só conseguia ser compilado na versão para Linux, na versão para Windows alguns erros ocorriam.

Basicamente eram estas as observações a serem feitas com relação a implementação do código. Algumas outras alternativas relacionadas a possíveis melhorias de desempenho também foram levantadas, como por exemplo, a estratégia de *merge-sort em disco*. Porém o fato das seqüências possuírem tamanhos variados tornava o método bem mais

complicado. Assim foi decidido rodar primeiramente os experimentos, e verificar se o método seria vantajoso, se fosse, estas alternativas seriam implementadas.

4.2.3 Resultados

4.2.3.1 Dados dos Fragmentos Gerados

Na Tabela 8 são exibidos, para cada versão da base de dados, os resultados relativos à distribuição das seqüências entre os fragmentos.

Tabela 8 - Distribuição das Seqüências entre os Fragmentos

F.		NAT		ORD		PRO	
		Numero bp	Num. Seq	Numero bp	Num. Seq	Numero bp	Num. Seq
2	1	1.027.999.063	335.233	1.027.867.519	5.441	1.027.614.471	287.427
	2	1.027.229.885	239.622	1.027.361.429	569.414	1.027.614.477	287.428
DP		384.589	47.806	253.045	281.987	3	1
4	1	513.866.824	184.679	513.994.976	2.249	513.807.237	143.712
	2	513.997.006	150.459	513.872.543	3.192	513.807.237	143.713
	3	513.999.592	125.330	513.997.179	31.622	513.807.237	143.715
	4	513.365.526	114.387	513.364.250	537.792	513.807.237	143.715
DP		260.611	27.026	260.685	227.827	0	1
8	1	256.999.922	88.589	256.840.783	940	256.903.618	71.849
	2	256.866.902	96.090	256.969.329	1.308	256.903.618	71.851
	3	256.967.205	60.610	256.860.613	1.485	256.903.618	71.859
	4	256.965.266	89.842	256.920.461	1.706	256.903.623	71.860
	5	256.999.575	88.323	256.966.423	2.366	256.903.617	71.860
	6	256.999.535	36.948	256.998.011	29.173	256.903.618	71.859
	7	256.832.973	44.984	256.999.237	114.544	256.903.618	71.859
	8	256.597.570	69.469	256.674.091	423.333	256.903.618	71.858
DP		130.159	21.020	103.126	137.790	2	4
16	1	128.999.903	29.477	128.780.084	359	128.451.809	35.915
	2	128.997.731	59.725	128.900.573	585	128.451.809	35.918
	3	128.926.398	37.572	128.972.510	636	128.451.809	35.927
	4	128.999.998	58.609	128.819.982	677	128.451.809	35.927
	5	128.999.503	24.625	128.861.596	721	128.451.809	35.928
	6	128.996.042	35.819	128.926.884	771	128.451.809	35.928
	7	128.999.117	57.126	128.898.022	824	128.451.809	35.929
	8	128.997.883	35.305	128.880.314	891	128.451.809	35.929
	9	128.999.334	50.870	128.910.062	1.027	128.451.809	35.929
	10	128.997.056	36.439	128.991.145	1.376	128.451.809	35.930
	11	128.836.162	16.014	128.995.801	5.032	128.451.809	35.931
	12	128.882.976	19.681	128.996.977	25.573	128.451.809	35.932
	13	128.998.634	26.886	128.997.761	45.716	128.451.809	35.932
	14	128.917.018	17.906	128.998.646	71.643	128.451.814	35.934
	15	128.998.593	24.144	128.999.244	115.401	128.451.808	35.933
	16	120.682.600	44.657	121.299.347	303.623	128.451.809	35.933
DP		2.006.611	14.160	1.847.983	176.141	1	5

Conforme foi suposto, a estratégia proposta – 'nt.prop' – gera a melhor distribuição em todas as configurações – 2, 4, 8 e 16 fragmentos – tanto para o número de seqüências quanto para o número de pares de base (pb) alocados em cada fragmento. Este fato pode ser verificado tanto pelo pequeno valor do desvio padrão (DP) apresentado na Tabela 8, quanto visualmente através dos gráficos da Figura 49.

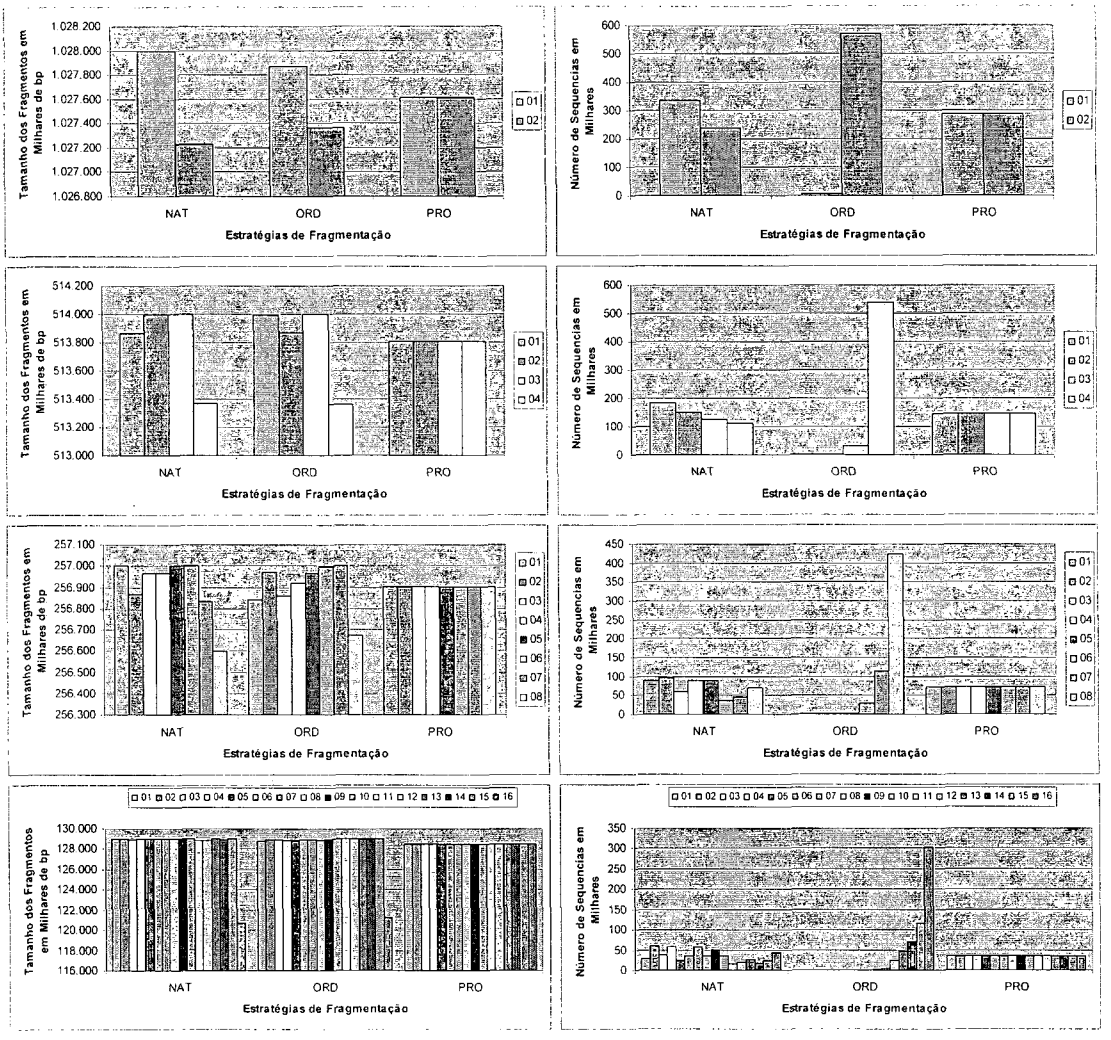

Figura 49 - Gráficos da Distribuição dos Dados entre os Fragmentos

Confirmando também as expectativas, a versão ordenada da base – 'nt.ord' – gerou, em todas as configurações, uma péssima distribuição do número de seqüências em cada fragmento, conforme indicam os gráficos e os elevados valores do desvio padrão. Pode também ser verificado que o último fragmento de cada configuração recebe o maior número de

seqüências, tal fato se deve a base de dados ter suas seqüências em ordem decrescente de tamanho.

Também pode ser observado, nas versões 'nt.nat' e 'nt.ord', que o último fragmento de cada configuração é sempre o de menor tamanho. Isto ocorre, pois o mpiformatdb utiliza o parâmetro '-v' do formatdb, para informar o tamanho desejado para os fragmentos. Como este parâmetro espera valores em milhões de caracteres, não é possível ser muito preciso, fazendo com que o último fragmento fique em geral com a 'sobra' da base de dados. Por esta razão, o mpiformatdb arredonda para cima o valor calculado para o tamanho dos fragmentos, pois caso contrário o formatdb poderá gerar um fragmento a mais da quantidade desejada. Apesar disso, devido ao tamanho da base de dados, os tamanhos dos fragmentos são aproximadamente iguais.

Na Tabela 9 é apresentado, em valores percentuais, a variação do número de seqüências e do número de pares de base em cada fragmento, em relação ao maior valor obtido para ambos em cada configuração.

Tabela 9 - Variação do Número de Seqüências e Pares de Base

FRAG		NAT		ORD	
		Max bp	Max Seq	Max bp	Max Seq
2	1	0,000%	0,000%	0,000%	**-99,044%**
	2	**-0,075%**	**-28,521%**	**-0,049%**	0,000%
4	1	-0,026%	0,000%	0,000%	**-99,582%**
	2	-0,001%	-18,529%	-0,024%	**-99,406%**
	3	0,000%	**-32,136%**	0,000%	**-94,120%**
	4	**-0,123%**	**-38,062%**	**-0,123%**	0,000%
8	1	0,000%	-7,806%	-0,062%	**-99,778%**
	2	-0,052%	0,000%	-0,012%	**-99,691%**
	3	-0,013%	**-36,924%**	-0,054%	**-99,649%**
	4	-0,013%	-6,502%	-0,031%	**-99,597%**
	5	0,000%	-8,083%	-0,013%	**-99,441%**
	6	0,000%	**-61,549%**	0,000%	**-93,109%**
	7	-0,065%	**-53,186%**	0,000%	**-72,942%**
	8	**-0,157%**	**-27,704%**	**-0,127%**	0,000%

FRAG	NAT		ORD	
	Max bp	Max Seq	Max bp	Max Seq
1	0,000%	**-50,645%**	-0,170%	**-99,882%**
2	-0,002%	0,000%	-0,076%	**-99,807%**
3	-0,057%	**-37,092%**	-0,021%	**-99,791%**
4	0,000%	-1,869%	-0,139%	**-99,777%**
5	0,000%	**-58,769%**	-0,107%	**-99,763%**
6	-0,003%	**-40,027%**	-0,056%	**-99,746%**
7	-0,001%	-4,352%	-0,078%	**-99,729%**
8	-0,002%	**-40,887%**	-0,092%	**-99,707%**
9	-0,001%	-14,826%	-0,069%	**-99,662%**
10	-0,002%	**-38,989%**	-0,006%	**-99,547%**
11	-0,127%	**-73,187%**	-0,003%	**-98,343%**
12	-0,091%	**-67,047%**	-0,002%	**-91,577%**
13	-0,001%	**-54,984%**	-0,001%	**-84,943%**
14	-0,064%	**-70,019%**	0,000%	**-76,404%**
15	-0,001%	**-59,575%**	0,000%	**-61,992%**
16	**-6,448%**	-25,229%	**-5,969%**	0,000%

Podemos verificar, nas configurações com 2, 4 e 8 fragmentos, que a variação do menor fragmento em relação ao maior é quase inexpressiva, não chegando a 0,2% (valores realçados na tabela). Já na

configuração com 16 fragmentos esta variação já é maior, chegando a 6,5%, mesmo assim continua não sendo muito expressiva. Esta maior variação nesta última configuração está relacionada ao fato dos fragmentos serem menores e assim acabam sofrendo uma maior influencia do parâmetro 'v' do formatdb.

Finalmente, pode ser observado que em ambas as versões, 'nt.nat' e 'nt.ord' a variação do número de seqüências entre os fragmentos é bem acentuada (valores realçados na tabela), apesar de ser bem mais expressiva, quase 100%, na versão ordenada.

4.2.3.2 Execução dos Experimentos

Ao contrário do que se esperava a execução dos experimentos não revelou ganho de desempenho para a estratégia de fragmentação proposta, conforme pode ser observado na Tabela 10 e nos gráficos da Figura 50.

Tabela 10 - Tempo das Rodadas

BASE	NUM. DE NOS E FRAGMENTOS			
	2	4	8	16
NT.NAT	52:34:39	26:55:56	13:55:20	07:25:48
NT.ORD	54:17:02	27:14:27	13:55:03	07:28:19
NT.PRO	52:25:38	26:57:51	13:49:47	07:21:19

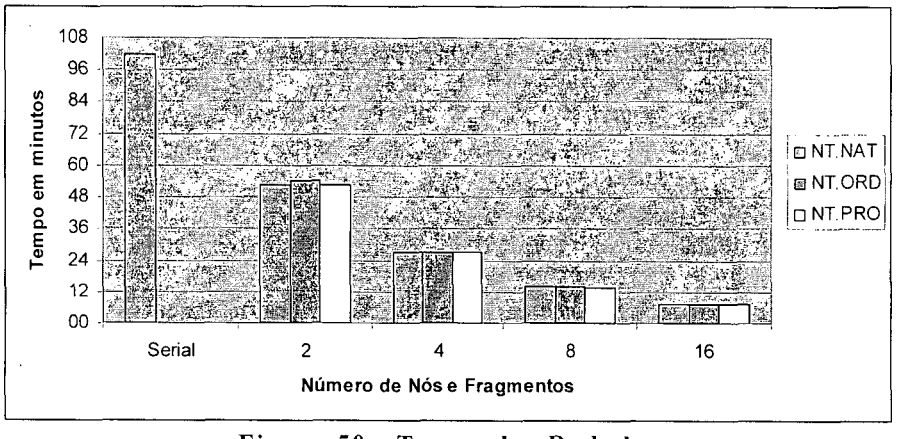

Figura 50 - Tempo das Rodadas

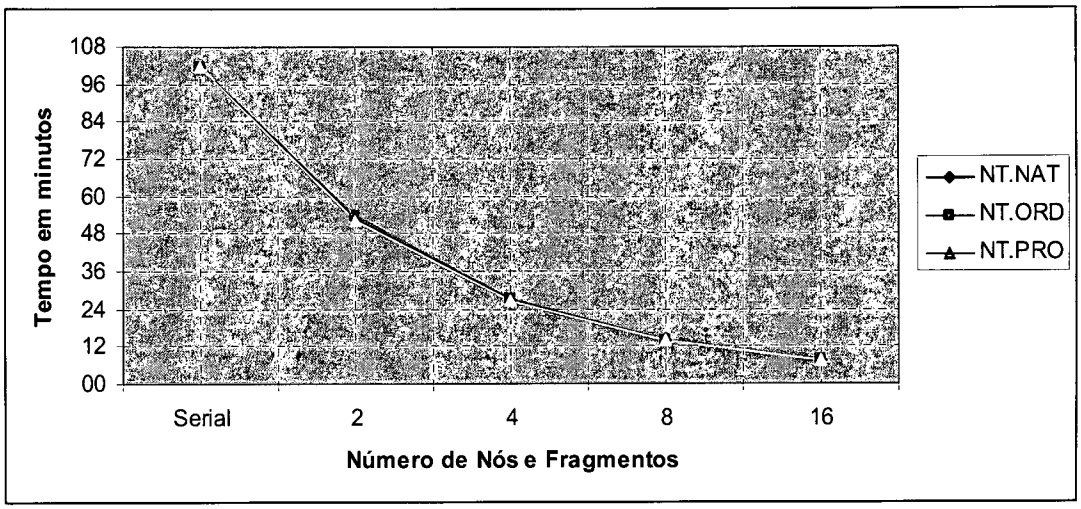

Figura 51 - Tempo das Rodadas - Linha de Tendência

Como os dados demonstram, praticamente não há diferença de tempo entre as estratégias, apesar da versão proposta ter sido a mais rápida em três (2, 8 e 16) das quatro configurações. Em apenas um caso a diferença de tempo chegou a 3,6%, o que não parece ser muito significativo. Em todos os outros casos a diferença não chega a 1,5%, como pode ser visto na Tabela 11, que apresenta em porcentagem a diferença de tempo entre cada execução e a execução mais rápida em cada configuração.

Tabela 11 - Percentual do Tempo Relativo a Execução mais Rápida

BASE	NUM. DE NÓS E FRAGMENTOS			
	2	4	8	16
NT.NAT	0,29%	0,00%	0,67%	1,02%
NT.ORD	3,54%	1,15%	0,64%	1,59%
NT.PRO	0,00%	0,12%	0,00%	0,00%

4.2.4 Análise dos Resultados

Os resultados obtidos na seção anterior indicam que o tempo necessário para realizar as buscas nos fragmentos, depende mais do tamanho destes do que da distribuição das seqüências em cada um. Assim, as pequenas diferenças de tempo encontradas estariam provavelmente relacionadas ao grau de similaridade entre as seqüências da consulta e da base de dados.

Para poder avaliar o fato da distribuição das seqüências da base de dados entre os fragmentos não ter influenciado no tempo de busca, duas análises adicionais foram efetuadas. Na primeira delas, foi realizado um estudo, baseado na descrição do algoritmo do BLAST, no qual é verificado a influência do tamanho das seqüências e de alguns parâmetros de execução do BLAST, na obtenção de um número maior de acertos na base de dados (fase um do BLAST); fator que poderia influenciar no tempo de execução do BLAST. Na segunda, é feita uma análise dos dados utilizados nos experimentos – seqüências da consulta, seqüências da base de dados e alinhamentos obtidos – para verificar a relação destes com as conclusões obtidas a partir da primeira análise feita. Assim, baseado nos dois estudos efetuados será apresentado a possível razão pelo qual não foi obtido ganho de desempenho com a proposta feita.

Conforme a descrição do algoritmo do BLAST, temos que uma das principais razões de sua velocidade é a redução do espaço de busca, que é alcançada fazendo a escolha das sementes e em seguida estendendo-as. Esta estratégia de certa forma confirma a tese proposta na seção 3.2.1 na qual o tamanho da seqüência teria influencia no número de acertos encontrados; e, assim, seqüências menores teriam uma probabilidade menor de fazer parte das fases dois e três do algoritmo do BLAST, levando-o a uma execução mais curta e por conseqüência mais rápida.

Para reavaliar a influencia do tamanho das seqüências na obtenção de um número maior de acertos – na primeira fase do BLAST – foi feito um estudo do número de palavras obtidas de uma base de dados, dado um tamanho de palavra W. Neste estudo o tamanho da base de dados é fixado de maneira que o seu número de seqüências (M) seja dado em função do tamanho de cada seqüência (T), que no caso é fixo para todas. Assim para uma base de dados com 10.000.000 de caracteres teríamos, por exemplo, a configuração de T=10 e M=1.000.000; T=100 e M=100.000; e assim por diante.

Para cada configuração obtida, foi calculado o número total de palavras encontradas na base de dados em função do valor de W. Este cálculo foi feito multiplicando-se o número de palavras de uma seqüência (n), pelo número de seqüências (M). O número de palavras de uma seqüência pode ser obtido através da seguinte equação: n = T–W+1. Esta equação corresponde ao número de deslocamentos possíveis de uma janela com tamanho W, sobre uma seqüência de tamanho T. Na Figura 60 este processo é exemplificado para uma seqüência de tamanho 8, nos casos em que W=3 e W=4. No primeiro caso, o número **n** de palavras obtidas é 6 (8 – 3 + 1) e no segundo 5 (8 – 4 + 1), confirmando a equação.

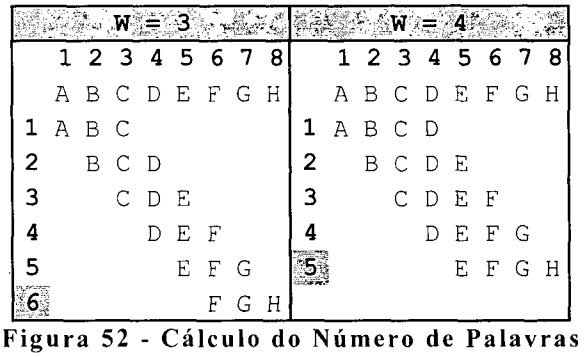

Figura 52 - Cálculo do Número de Palavras

Na Tabela 12 é apresentado o número de palavras encontradas na base de dados em função dos valores de W, M, T, onde o valor de M vezes T é constante e igual a 10.000.000. Foram escolhidos para W, valores em potências de dois, variando de 2 a 1024. Além destes, foram incluídos os valores 3 e 11 que são utilizados por padrão pelo BLAST.

Tabela 12 - Relação do Número de Palavras em Função de W, T e M

T	1.000.000	100.000	10.000	1.000	500	250	100	50	10
M	10	100	1.000	10.000	20.000	40.000	100.000	200.000	1.000.000
W	Número de Palavras em Cada Configuração de M, T e W, onde D = M x T = 10.000.000								
2	9.999.990	9.999.900	9.999.000	9.990.000	9.980.000	9.960.000	9.900.000	9.800.000	9.000.000
3	9.999.980	9.999.800	9.998.000	9.980.000	9.960.000	9.920.000	9.800.000	9.600.000	8.000.000
4	9.999.970	9.999.700	9.997.000	9.970.000	9.940.000	9.880.000	9.700.000	9.400.000	7.000.000
8	9.999.930	9.999.300	9.993.000	9.930.000	9.860.000	9.720.000	9.300.000	8.600.000	3.000.000
11	9.999.900	9.999.000	9.990.000	9.900.000	9.800.000	9.600.000	9.000.000	8.000.000	0
16	9.999.850	9.998.500	9.985.000	9.850.000	9.700.000	9.400.000	8.500.000	7.000.000	0
32	9.999.690	9.996.900	9.969.000	9.690.000	9.380.000	8.760.000	6.900.000	3.800.000	0
64	9.999.370	9.993.700	9.937.000	9.370.000	8.740.000	7.480.000	3.700.000	0	0
128	9.998.730	9.987.300	9.873.000	8.730.000	7.460.000	4.920.000	0	0	0
256	9.997.450	9.974.500	9.745.000	7.450.000	4.900.000	0	0	0	0
512	9.994.890	9.948.900	9.489.000	4.890.000	0	0	0	0	0
1024	9.989.770	9.897.700	8.977.000	0	0	0	0	0	0

Conforme pode ser observado, o número de palavras não sofre uma grande alteração quando o valor de W é pequeno em relação a T. Por exemplo, para W=2, T=1.000.000 e M=10 temos 9.999.990 palavras; já para W=2, T=100.000 e M=100 temos 9.999.900 palavras, ou seja apenas 90 palavras a menos, uma variação que não chega a 0,001%. Esta variação pode ser melhor visualizada através do gráfico da Figura 53. Neste gráfico, seqüências com tamanho 100.000 e 1.000.000 praticamente não sofrem alteração com a variação de W. Já as seqüências com tamanhos entre 250 e 1000, sofrem uma variação significativa apenas quando W varia de 32 em diante.

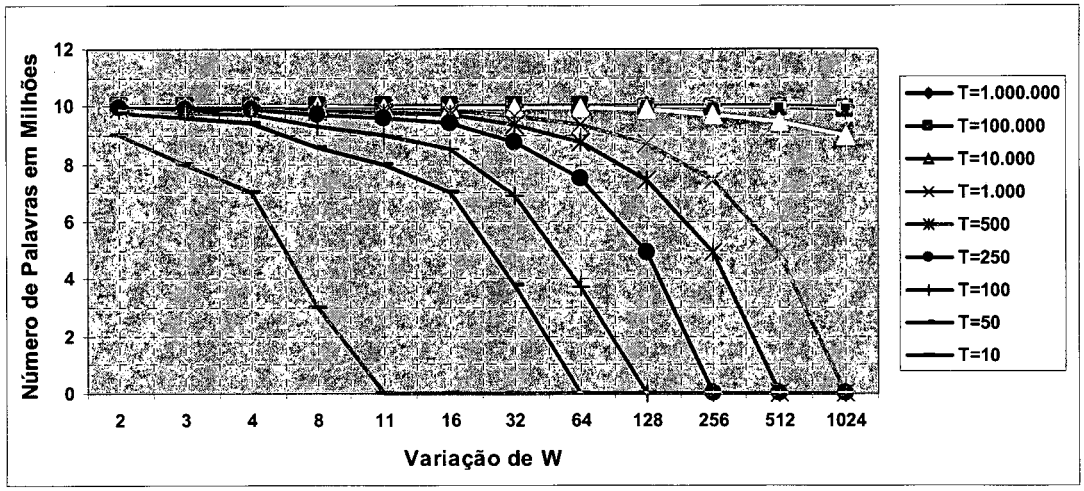

Figura 53 - Variação do Número de Palavras em Função de W, T e M

Uma outra análise importante é verificar o comportamento do gráfico para os valores de W que são utilizados por padrão pelo BLAST – 11 para o BLASTn e 3 para os demais. Nota-se que para W=3 apenas as seqüências de até 10 caracteres sofreriam uma variação relevante, já para W=11 o mesmo ocorreria apenas para as seqüências com até 100 caracteres.

Portanto, este estudo demonstrou que dependendo dos valores de W e T, não haverá diferença no número de palavras obtidas, desde que o tamanho dos fragmentos seja aproximadamente o mesmo e W seja relativamente pequeno em relação a T. Assim a tese anteriormente proposta não teria efeito nestes casos. Para poder avaliar se seria este o caso dos experimentos aqui analisados, foi realizado um levantamento estatístico das seqüências da consulta e da base de dados, bem como dos alinhamentos obtidos. Esta análise teve por objetivo obter uma idéia mais precisa de como as seqüências estão distribuídas em função de seus tamanhos e das características dos alinhamentos obtidos. Baseado nesta análise será verificado se existe na base de dados um número significativo de seqüências que poderiam sofrer influência do valor utilizado para W (três) nos experimentos executados e se seria viável aumentar este valor.

Na Figura 54 é apresentado um levantamento estatístico a respeito do comprimento dos alinhamentos obtidos com as seqüências da consulta utilizada. Como pode ser observado cerca de 66% dos alinhamentos possui até 40 caracteres de comprimento, aumentando para 92% os que possuem até 80 caracteres, ou seja quase a totalidade deles.

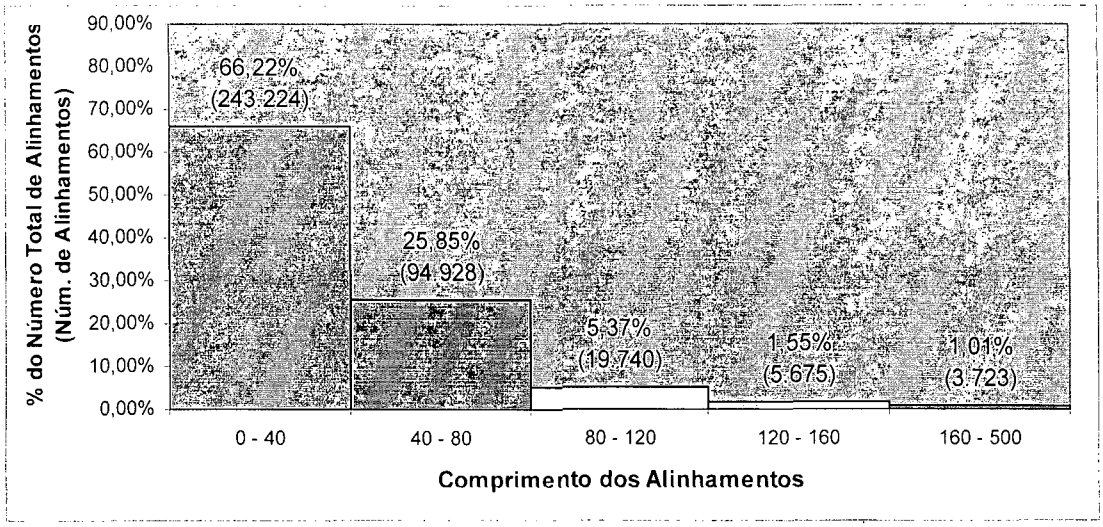

Figura 54 - Estatística dos Alinhamentos

Como os alinhamentos obtidos foram realizados entre seqüências de aminoácidos, as estatísticas das seqüências da consulta e da base de dados foram realizadas considerando um terço do comprimento de cada seqüência.

O levantamento estatístico das seqüências da consulta foi abordado de duas formas. Na primeira delas, as seqüências foram agrupadas por tamanho e, para cada grupo formado, foi calculado o número de seqüências presente e o seu peso (em termos percentuais) em relação ao número total de seqüências. Na segunda, verificou-se o peso (também em termos percentuais) do tamanho de cada grupo – somando o tamanho de suas seqüências – em relação ao tamanho total do arquivo de consultas. Nas Figura 55 e Figura 56 são apresentados os resultados da primeira e segunda abordagem, respectivamente.

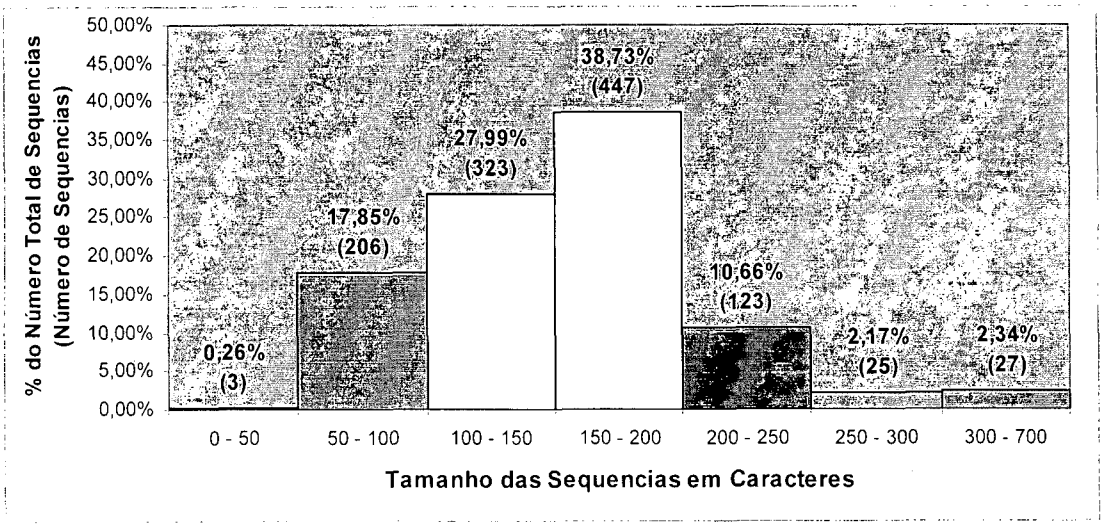

Figura 55 - Seqüências da Consulta Agrupadas por Tamanho

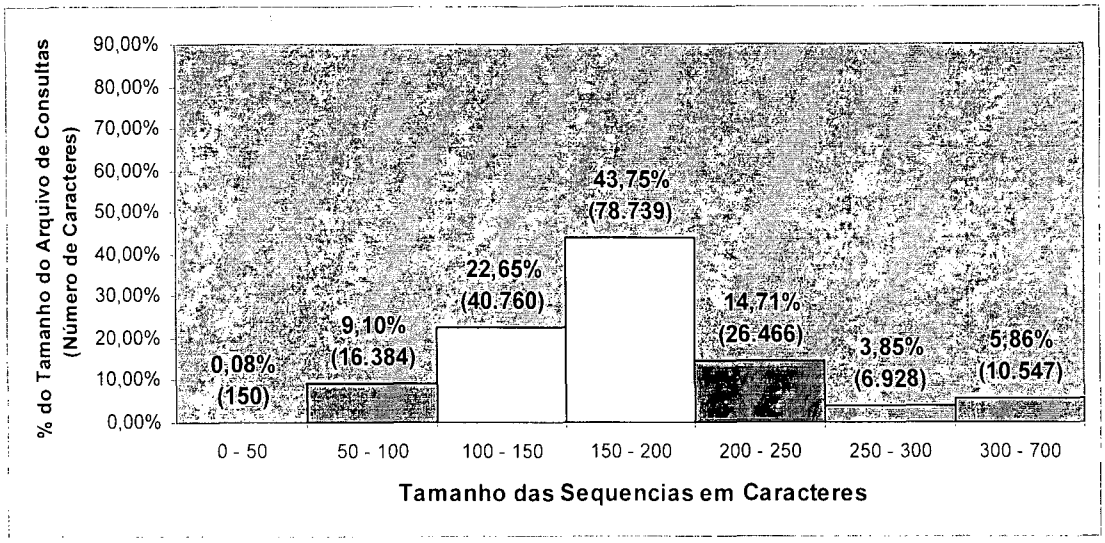

Figura 56 - Peso dos Grupos em Relação ao Tamanho Total da Consulta

Conforme pode ser observado no gráfico, a maioria das seqüências, cerca de 95%, possui entre 50 e 250 caracteres de comprimento, e destas, 70% possui entre 100 e 200 caracteres. Já pelo gráfico da Figura 56 pode ser verificado que o peso dos três primeiros grupos decai bastante em relação ao primeiro gráfico. Com os três últimos grupos ocorre o contrário. O único grupo estável foi o quarto, o das seqüências com comprimento entre 100 e 150 caracteres.

Um levantamento semelhante ao anterior foi realizado para as seqüências da base de dados. Aqui houve uma grande inversão de uma abordagem para a outra. Conforme pode ser observado no gráfico da

Figura 57, o grupo um – seqüências com até 5.000 caracteres – engloba quase todas as seqüências, com cerca de 98% delas, porém conforme pode ser observado no gráfico da Figura 58, sua relevância em relação ao tamanho total da base de dados é bastante reduzida, caindo para 31%. O processo inverso ocorre com o quinto grupo – seqüências com mais de 20.000 caracteres – que sofre um salto de 1,4% para 65%.

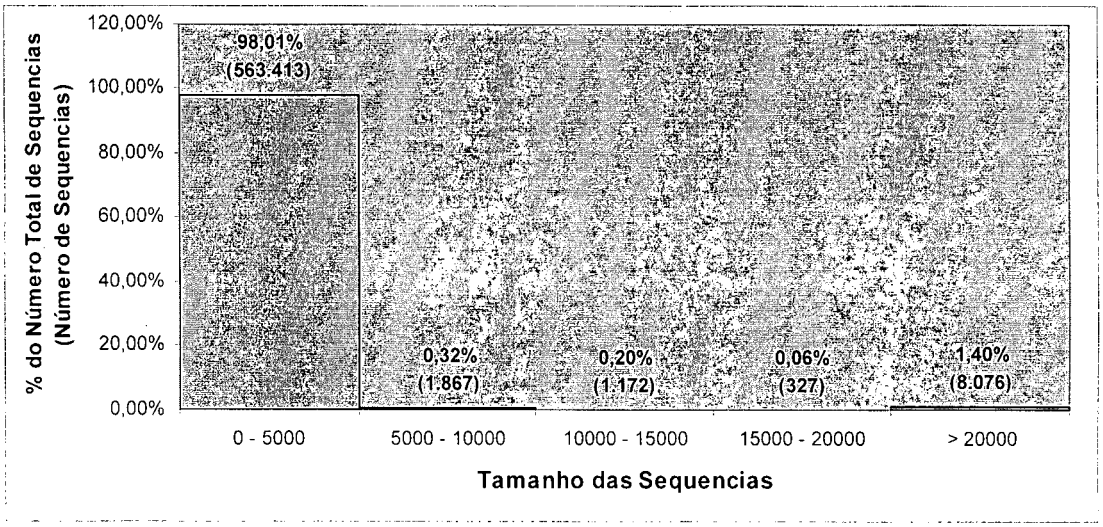

Figura 57 - Seqüências da Base de Dados Agrupadas por Tamanho

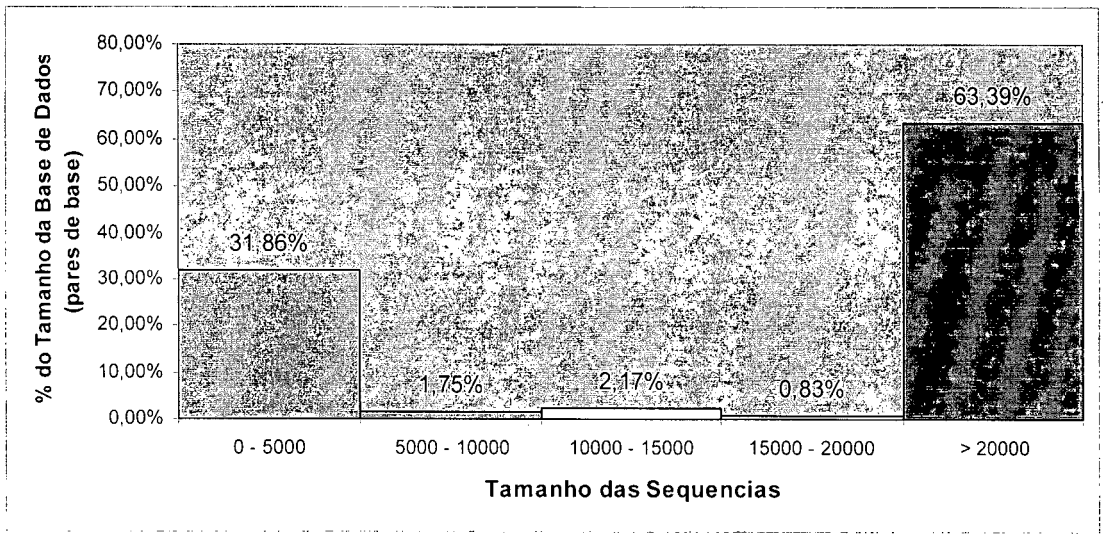

Figura 58 - Peso dos Grupos em Relação ao Tamanho Total da Base de Dados

Devido a grande quantidade de seqüências no grupo com ate 5.000 caracteres, um segundo levantamento foi feito para avaliá-lo em separado. Foram criados 20 grupos, de 50 caracteres cada, para agrupar

as seqüências com até 1.000 caracteres, e mais um grupo, o 21, para as seqüências entre 1.000 e 5.000 caracteres.

Seguindo a tendência dos gráficos anteriores, as menores seqüências possuem um peso maior na quantidade de que no tamanho total. Os grupos de 1 a 6 – seqüências com até 300 caracteres – seguem esta tendência, como pode ser observado nos gráficos das Figura 59 e Figura 60, e através da Tabela 13. As seqüências maiores por sua vez, apesar de em menor número possuem um peso maior em relação ao tamanho total, como é o caso do grupo 21 – seqüências com mais de 1000 caracteres – que sofre um salto de 7% para 30%.

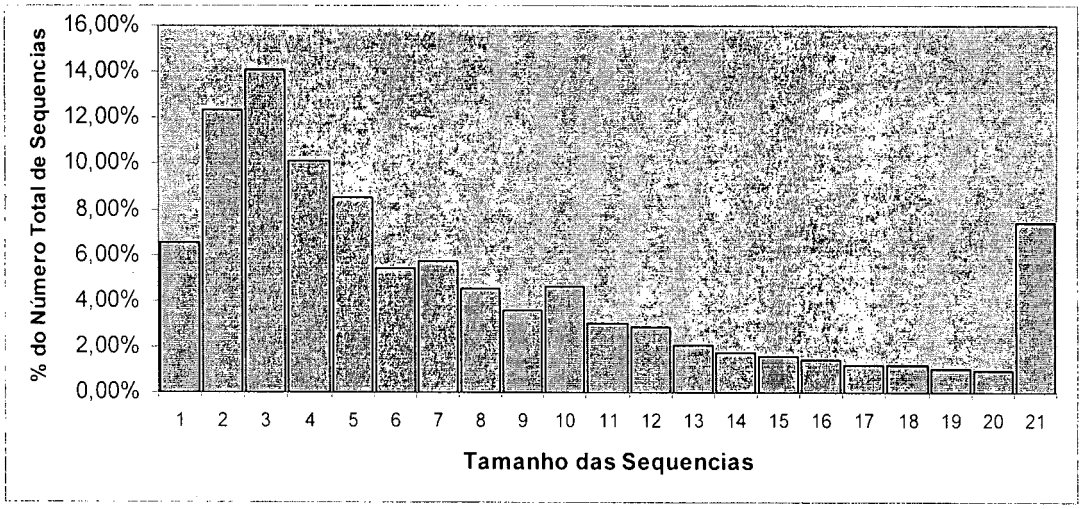

Figura 59 - Seqüências de ate 5.000 caracteres Agrupadas por Tamanho

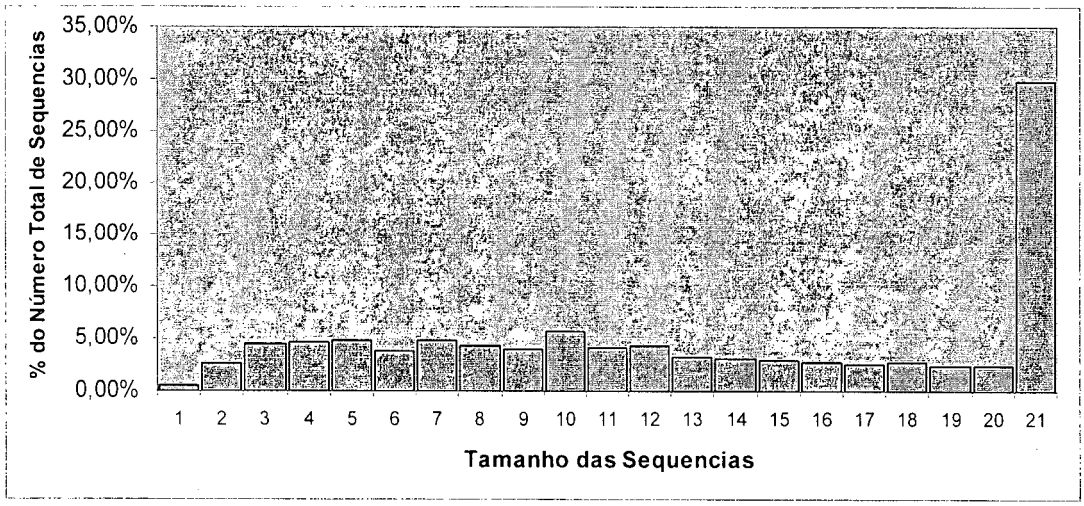

Figura 60 - Peso dos Grupos em Relação ao Tamanho Total

Tabela 13 - Percentual dos Grupos

	Faixa	% Num Sequecias	% Tamanho		Faixa	% Num Sequecias	% Tamanho
1	0 - 50	6,53%	0,48%	11	500 - 550	3,05%	4,11%
2	50 - 100	12,37%	2,51%	12	550 - 600	2,85%	4,25%
3	100 - 150	14,09%	4,52%	13	600 - 650	2,06%	3,32%
4	150 - 200	10,12%	4,58%	14	650 - 700	1,73%	3,02%
5	200 - 250	8,54%	4,90%	15	700 - 750	1,59%	2,98%
6	250 - 300	5,44%	3,87%	16	750 - 800	1,40%	2,81%
7	300 - 350	5,69%	4,79%	17	800 - 850	1,23%	2,62%
8	350 - 400	4,54%	4,39%	18	850 - 900	1,19%	2,69%
9	400 - 450	3,61%	3,97%	19	900 - 950	1,00%	2,39%
10	450 - 500	4,61%	5,68%	20	950 - 1000	0,93%	2,35%
				21	> 1000	7,42%	29,79%

Como pode ser constatado, apenas 0,15% (0,48% x 31,86%) da base de dados é composto por seqüências com ate 50 caracteres. Já as seqüências com até 100 caracteres corresponderiam a apenas 0,95% da base. Conforme o estudo do número de palavras obtidas dado o tamanho da seqüência e o tamanho W das palavras (Figura 53), temos que apenas 0,15% da base de dados seria afetada por um valor de W=3, que é o valor utilizado pelo tBLASTx.

A utilização de valores maiores para W, é permitida pelo BLAST, porém dado que 95% das seqüências da consulta utilizada possuem até 250 caracteres, um aumento de W não parece ser viável, pois o algoritmo provavelmente perderia em sensibilidade e assim não retornaria resultados para a consulta. Conforme os resultados estatísticos dos alinhamentos, temos que 92% deles possuem até 80 caracteres. Como o BLAST constrói alinhamentos com inserção de lacunas, grandes valores para W dificultariam muito a obtenção de acertos. Em [53], um estudo feito pelo NCBI recomenda o uso de W=50 para seqüências grandes de consulta, da ordem de 250Kb ou aproximadamente 250.000 caracteres de comprimento. Assim, apesar de ser possível aumentar o valor de W, este somente deve ser feito para seqüências de consulta grandes.

Portanto baseado nos levantamentos feitos a respeito da base de dados, das seqüências da consulta e para o valor de W utilizado nos experimentos, qualquer estratégia de fragmentação que gerasse

fragmentos de aproximadamente o mesmo tamanho seria suficiente para se alcançar um bom balanceamento de carga no mpiBLAST.

Apesar da análise da base de dados do 'nt' não ter sido realizada sobre sua totalidade, ou seja, em apenas 2GB da base original, uma análise idêntica (Apêndice A) foi realizada sobre a base de dados completa e apresentou praticamente a mesma distribuição com diferenças praticamente desprezíveis. Assim, os resultados obtidos podem ser generalizados para a base de dados completa.

4.3 Estratégia com fragmentação da cosulta

4.3.1 Método de Avaliação e Detalhes de Implementação

A avaliação da segunda estratégia proposta nesta dissertação foi facilitada, pois parte das implementações para a avaliação da primeira proposta foram reaproveitadas para a avaliação desta. A base de dados utilizada para os testes foi a base reduzida do nt. O tipo de busca BLAST utilizado foi inicialmente o tBLASTx e posteriormente o BLASTn. Devido a alguns problemas com o "cluster" do NACAD, as rodadas ficaram restritas ao uso de até 8 nós.

Para avaliar o peso do processo de consolidação dos resultados no tempo total de execução, foram criadas as seguintes configurações:

- I) 8 nós, 8 fragmentos e uma execução do mpiBLAST sem fragmentação da consulta;
- II) 8 nós, 4 fragmentos e duas execuções do mpiBLAST com a consulta fragmentada em duas partes;
- III) 8 nós, 2 fragmentos e quatro execuções mpiBLAST com a consulta fragmentada em quatro partes.

O processo de fragmentação das consultas utilizou o algoritmo proposto na seção anterior, para gerar fragmentos de aproximadamente o mesmo tamanho.

A execução da configuração I consistiu apenas em chamar o mpiBLAST por sua forma tradicional. Já a execução dos experimentos com fragmentação da consulta – configurações II e III – seguiu as seguintes etapas:

- fragmentar a consulta;
- fazer as chamadas ao mpiBLAST para cada fragmento da consulta;
- concatenar os resultados.

Devido a simplicidade deste processo, apenas um script em Shell UNIX [54] foi gerado para executar cada um destes processos. Para cada uma das chamadas ao mpiBLAST foi submetido uma execução em separado ao sistema de filas PBS. Esta estratégia foi necessária para que a etapa de consolidação fosse iniciada, somente após o termino das execuções do mpiBLAST. Assim, um processo de consolidação também foi submetido ao PBS, contendo como requisitos para sua execução, o término dos processos do mpiBLAST. Este controle é feito automaticamente pelo PBS.

4.3.2 Resultados e Análise

Na Tabela 14 são apresentados os tempos coletados para cada configuração utilizando a busca tBLASTx. Como o tempo para fragmentação da consulta e concatenação dos resultados não chegou a dois segundos, estes foram incorporados aos resultados apresentados sem serem explicitamente descriminados. Nesta tabela são apresentados os tempos individuais, quando for o caso, referentes a execução do mpiBLAST para cada fragmento da consulta. Os valores destacados são os valores máximos de cada configuração, que determinam o tempo de resposta. Conforme pode ser observado, a diferença de tempo na busca, entre os fragmentos de uma mesma configuração é bem pequena, sendo

de 0,5% na configuração II em relação ao maior valor obtido, e de 2,5% na configuração III.

Tabela 14 - Tempos das Execuções

Num. Fragmentos da Consulta	I	II	III
	13:55:20	13:05:39	12:42:50
1	13:55:20	13:05:39	12:23:27
2		13:01:57	12:37:38
3			12:40:17
4			12:42:50

Na Figura 61 é apresentado um gráfico com o ganho de tempo entre as configurações. A configuração II tem um ganho de aproximadamente 6% em relação a configuração I e a configuração III um ganho de aproximadamente 8,5% e 3% em relação as configurações I e II, respectivamente.

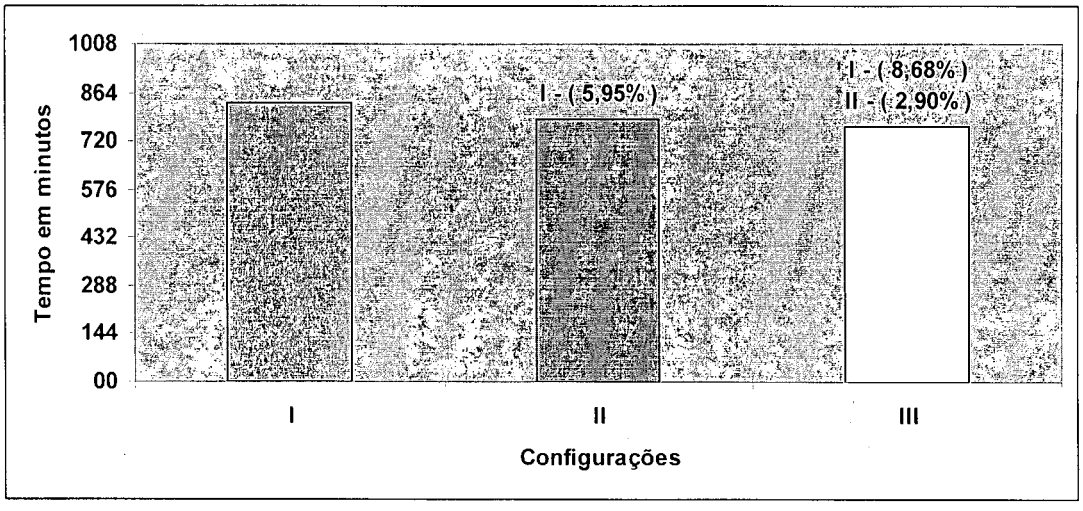

Figura 61 - Ganho de Tempo com Fragmentação da Consulta

Por ser o tipo de consulta BLAST mais demorado, o tempo da fase de consolidação dos resultados pode não ser muito significativo em relação ao tempo de execução do tBLASTx. Para fazer esta verificação, os experimentos foram repetidos utilizando a busca entre nucleotídeos, BLASTn.

Na Tabela 15 são exibidos os tempos obtidos. Como ocorreu na execução anterior, os fragmentos levaram aproximadamente o mesmo tempo de busca. Conforme era esperado, o peso da etapa de consolidação

em relação ao tempo total de execução foi bem mais expressivo para o BLASTn.

Tabela 15 - Tempos das Execuções com o BLASTn

Num. Fragmentos da Consulta	I	II	III
	00:44:52	00:36:36	00:32:49
1	00:44:52	00:36:32	00:32:32
2		00:36:36	00:32:46
3			00:32:40
4			00:32:49

Na Figura 62 é apresentado um gráfico com o ganho de tempo entre as configurações. Conforme pode ser observado, a configuração II teve um ganho de aproximadamente 18,5% em relação a configuração I e a configuração III um ganho de aproximadamente 27% e 10,34% em relação as configurações I e II, respectivamente

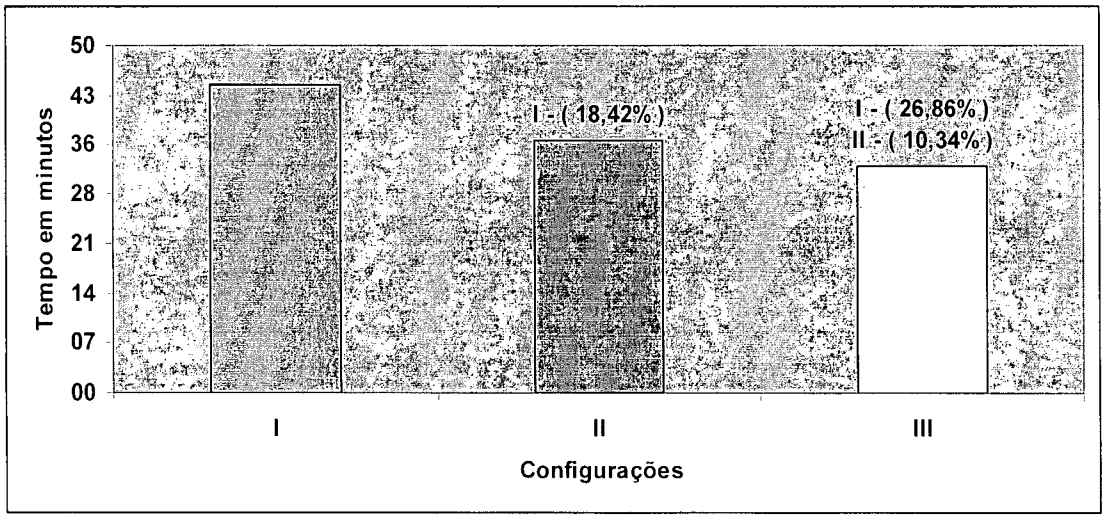

Figura 62 - Ganho de Tempo com Fragmentação da Consulta no BLASTn

Foi feito uma última análise para a busca BLASTn variando o tamanho do arquivo de consultas. Assim, foram criados mais dois arquivos, o primeiro contendo o dobro, e o segundo o quádruplo das seqüências do arquivo original. Na Tabela 16 são apresentados as tomadas de tempo para estas configurações.

Tabela 16 - Medidas de Tempo para a Variação da Consulta

CONSULTA	I	II	III
x 1	00:44:52	00:36:36	00:32:49
x 2	01:32:21	01:14:33	01:06:27
x 4	03:10:57	02:30:25	02:12:45

A variação de tempo para cada consulta pode ser melhor visualizada através do gráfico da Figura 64, nele pode ser observado que o ganho de desempenho na medida em que o tamanho da consulta aumenta é praticamente estável, tendo uma pequena tendência de aumento.

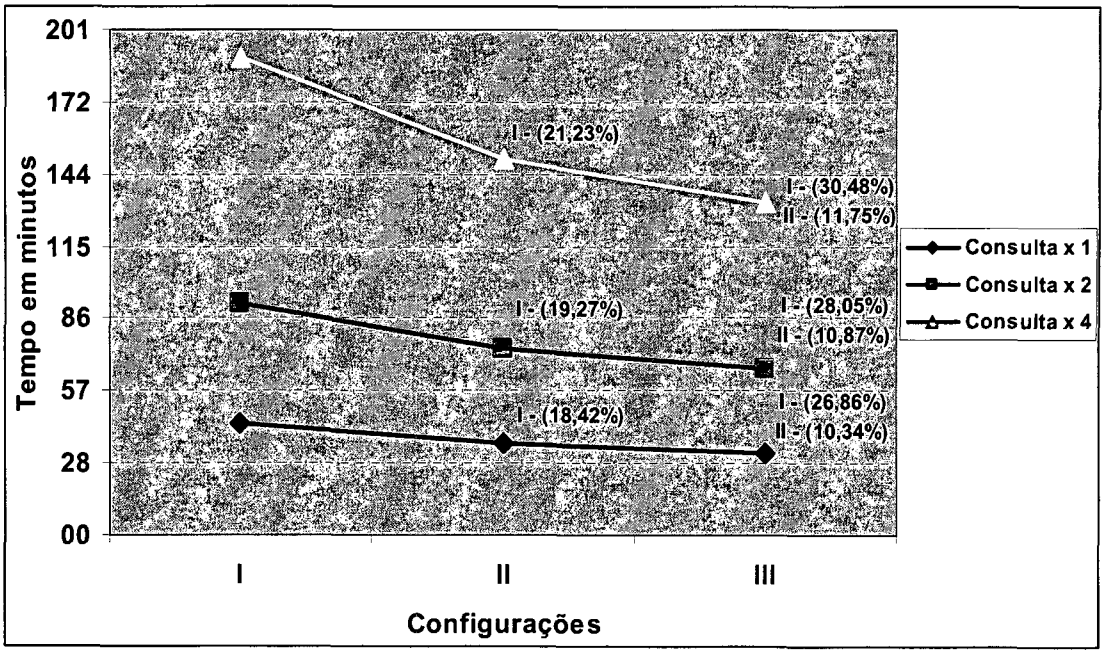

Figura 63 - Variação do Tempo em Relação ao Tamanho da Consulta

5 Conclusões

Nesta dissertação foi apresentada uma ferramenta de auxílio ao bioinformata para execuções do BLAST que requerem alto desempenho com uma boa relação custo/benefício. Foi realizada uma análise do mpiBLAST, uma ferramenta de software livre que implementa a execução em paralelo do BLAST através da fragmentação da base de dados. Inicialmente, foi feito um levantamento dos principais pontos em suas características de execução, que poderiam ser otimizados através da aplicação das estratégias de paralelismo já existentes. Esta avaliação procurou levar em conta as principais características de execução do BLAST, bem como os aspectos biológicos associados. Diversos experimentos foram realizados com o mpiBLAST e utilizando um "cluster" de PC sem hardware especial. Para a avaliação foram utilizadas seqüências de consultas de dados reais de pesquisa em biologia molecular para que os resultados obtidos refletissem um padrão real de uso do BLAST.

O estudo detectou dois aspectos principais que poderiam em ambos os casos levar a um desbalanceamento na distribuição da carga de trabalho:

- A estratégia utilizada para geração dos fragmentos da base de dados;
- A centralização da etapa de consolidação e composição dos resultados em um único nó de processamento.

No primeiro caso foi questionado se o fato de gerar fragmentos com aproximadamente o mesmo tamanho seria suficiente para se alcançar um bom balanceamento de carga. Foi levantada a hipótese de que além do tamanho, era também necessário fazer uma distribuição mais balanceada das seqüências grandes, médias e pequenas entre os fragmentos, para que o tempo de busca em cada um fosse aproximadamente o mesmo. Os testes e as análises realizadas mostraram

que, devido às características das seqüências de consultas (em sua maioria pequenas) e das seqüências da base de dados (em sua maioria grandes) qualquer estratégia de fragmentação que gerasse fragmentos de aproximadamente o mesmo tamanho seria suficiente para se alcançar um bom balanceamento de carga no mpiBLAST.

No segundo caso, foi levantada a hipótese de que a etapa de consolidação dos resultados pudesse ter um peso considerável no tempo total de execução do sistema. Como esta etapa era centralizada em um único nó de processamento, foi proposta uma estratégia que dividiria esta tarefa entre os outros nós. Assim, obteve-se um modelo misto, com fragmentação da consulta e da base de dados. Conforme os testes realizados demonstraram, este realmente era um ponto de estrangulamento do sistema e o novo modelo gerou ganho de desempenho, principalmente para o tipo de busca BLASTn.

Podemos considerar como uma importante contribuição deste trabalho, o estudo realizado do BLAST com relação ao seu funcionamento mediante as características da base de dados, das seqüências da consulta e de seus parâmetros de execução; bem como a influência disto em estratégias paralelas. Na maioria dos trabalhos publicados, apenas o tamanho dos fragmentos da base de dados foi considerado relevante para obtenção de um bom balanceamento de carga e em nenhum momento foi encontrado questionamentos a respeito da influência das características das seqüências no tempo de execução do algoritmo. Assim, pôde-se constatar, por exemplo, que para consultas com seqüências pequenas e para bases de dados com seqüências grandes, somente o tamanho dos fragmentos pode influenciar no tempo de resposta, o que de imediato não parece ser óbvio.

Outra contribuição importante foi o modelo proposto com fragmentação mista, pois além de fornecer um ganho de desempenho, a sua implementação é praticamente direta, podendo ser facilmente adotada

a qualquer projeto que utilize o mpiBLAST ou estratégia semelhante a esta.

Nos experimentos realizados observou-se que durante a execução do BLAST, o consumo de memória crescia gradativamente até chegar a um determinado limite, que dependendo do tamanho da base de dados, poderia ser a quantidade de memória disponível no sistema. Em seguida o uso de memória caía drasticamente, reiniciando o processo. Este comportamento confirma o padrão de execução do BLAST (Figura 64) conforme descrito em [40].

```
For each Query Sequence s {
  For each Database D_i {
    For each sequence d in D_i {
      Compare s to d using BLAST;
      Update aggregate statistics}}
    Report results for sequence s}
```

Figura 64 - Padrão de Execução do BLAST [40]

Para bases de dados grandes o suficiente, de forma que não caibam inteiramente em memória, esta estratégia fará com que as seqüências sejam lidas do disco para a memória, para cada seqüência de consulta. Seria interessante então inverter este processo e fazer com que as seqüências de consulta fossem para o laço mais interno, visto que são bem menores e caberiam facilmente em memória. Assim, uma seqüência da base de dados seria alinhada com todas as seqüências da consulta, e uma vez terminado este processo esta não seria mais necessária, diminuindo assim a quantidade de acesso a disco. Tal estratégia irá exigir profundas alterações na implementação atual do algoritmo do BLAST e assim sua viabilidade teria que ser avaliada.

Uma outra estratégia que poderia ser interessante seria armazenar as bases de dados de nucleotídeos já traduzidas em aminoácidos, e assim evitar esta etapa nas consultas que exigem a tradução. Esta estratégia também permite uma aplicação direta de estratégias de paralelismo.

Conforme descrito na seção 4.1 o tBLASTx efetua o equivalente a 36 buscas do tipo BLASTp. Fazendo a tradução da base de dados e da consulta em arquivos separados, poderia ser possível executar as 36 buscas em paralelo, bastando em seguida fazer a consolidação e composição dos resultados, o que é bem semelhante às estratégias já utilizadas.

Referencias Bibliográficas

[1] HEATH, L. S.,RAMAKRISHNAN, N., 2002, "The Emerging Landscape of Bioinformatics Software Systems", *IEEE Computer*, pp. 41-45

[2] ALTSCHUL, S. F., MADDEN, T. L., SCHÄFFER, A. A., et al, 1997, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs", *Nucleic Acids Research*, v. 25, n. 17, pp. 3389-3402.

[3] ALTSCHUL, S. F., GISH, W., MILLER, W., et al, 1990, "Basic Local Alignment Search Tool", *Journal of Molecular Biology*, v. 215, n. 3, pp. 403-410.

[4] DARLING, A. E., CAREY, L., FENG, W., 2003, "The Design, Implementation, and Evaluation of mpiBLAST".

[5] NCBI-BLAST. 2004, Disponível em: http://www.ncbi.nlm.nih.gov/BLAST/

[6] DÁVILA, A. M. R., BERRIMAN, M., GRISARD, E. C., et al., 2004, "The BiowebDB Consortium: an integrative Bioinformatics approach for comparative genomics of Kinetoplastida", Angra dos Reis, Rio de Janeiro, Brazil.

[7] CAVALCANTI, M. C., BAIÃO, F., RÖSSLE, S. C., et al, 2003, "Structural Genomic Workflows Supported by Web Services", *In Proceedings of Workshop on Biological Data Management DEXA' 03*, pp. 45-49.

[8] TEKAIA, F., 2005, "Bioinformatics and Genome Data Analysis course". Disponível em: http://www.pasteur.fr/~tekaia/BGDA.html, http://www.pasteur.fr/~tekaia/BGDA/BGDA_Introduction.ppt.

[9] SETÚBAL, J. C.,MEIDANIS, J., 1994, *Uma Introdução a Biologia Computacional*. Recife, Brasil, Biblioteca de Informática da Universidade Federal de Pernambuco.

[10] SETÚBAL, J. C.,MEIDANIS, J., 1997, *Introduction to Computational Molecular Biology*. 1 ed. Boston, USA, PWS Publishing.

[11] KANEHISA, M., 1999, *Post-Genome Informatics*. New York, Oxford University Press.

[12] ALBERTS, B., JOHNSON, A., LEWIS, J., et al, 2002, *Molecular Biology of the Cell*. 4 ed. New York, USA, Garland Science.

[13] HARTL, D. L.,JONES, E. W., 2004, *Genetics: Analysis of Genes and Genomes*. 6 ed. Massachusetts, USA, Jones & Bartlett Publishers.

[14] CLOTE, P.,BACKOFEN, R., 2000, *Computational Molecular Biology - An Introduction*. New York, USA, John Wiley & Sons.

[15] BEDELL, J., KORF, I., YANDELL, M., 2003, *BLAST*. Sebastopol, California, USA, O'Reilly & Associates, Inc.

[16] JUNIOR, S. A. D. C., 2003, *Sequence Alignment Algorithms*. Tese de M.Sc., University of London, London, England.

[17] NEEDLEMAN, S.,WUNSCH, C., 1970, "A General Method Applicable to the Search for Similarities in the Amino Acid Sequences of Two Proteins", *Journal of Molecular Biology*, v. 48, pp. 444-453.

[18] SMITH T.F.,WATERMAN M.S., 1981, "Identification of Common Molecular Subsequences", *Journal of Molecular Biology*, v. 147, n. 1, pp. 195-197.

[19] WILBUR, W.,LIPMAN, D. J., 1983, "Rapid Similarity Searches of Nucleic Acid and Protein Data Banks", *In Proceedings of the National Academy of Sciences*, v. 80, pp. 726-730.

[20] WU-BLAST. 2004, Disponível em: http://blast.wustl.edu/

[21] LEON, D.,MARKEL, S., 2003, *Sequence Analysis in a Nutshell*. Sebastopol, California, USA, O'Reilly & Associates, Inc.

[22] KARLIN, S.,ALTSCHUL, S. F., 1990, "Methods For Assessing the Statistical Significance of Molecular Sequence Features by Using General Scoring Schemes", *Proceedings of the National Academy of Sciences*, v. 87, pp. 2264-2268.

[23] DATE, C. J., 1986, *An Introduction to Database Systems*. 4 ed. Addison Wesley.

[24] BENSON, D. A., KARSCH-MIZRACHI, I., LIPMAN, D. J., et al, 2004, "GenBank: update", *Nucleic Acids Research*, v. 32, n. Database Issue.

[25] GENBANK, 2005, "GenBank Statistics". Disponível em: http://www.ncbi.nih.gov/Genbank/, http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html.

[26] ÖZSU, M. T.,VALDURIEZ, P., 1999, *Principles of Distributed Database Systems*. New Jersey, USA, Prentice Hall.

[27] ELMASRI, R.,NAVATHE, S. B., 2000, *Fundamentals of Database Systems*. 3 ed. Vancouver, Canada, Addison Wesley.

[28] BUYYA, R., 1999, *High Performance Cluster Computing: Architectures and Systems*. 1 ed. New Jersey, USA, Prentice Hall PTR.

[29] ZHU, Y., JIANG, H., QIN, X., et al, 2003, "A Case Study of Parallel I/O for Biological Sequence Search on Linux Clusters", pp. 308-315, Hong Kong.

[30] SINGH, R. K., DETTLOFF, W. D., CHI, V. L., et al, 1996, "BioSCAN: A Dynamically Reconfigurable Systolic Array for Biosequence Analysis", pp. 22-24.

[31] RABAEY, J. M., CHANDRAKASAN, A., NIKOLIC, B., 2002, *Digital Integrated Circuits*. 2 ed. New Jersey, USA, Prentice Hall.

[32] LAVENIER, D., 1998, "Speeding up genome computation with a systolic accelerator", *SIAM news*, v. 31, n. 8.

[33] TimeLogic. 2004, Disponível em: http://www.timelogic.com/downloads/decypher_overview.pdf

[34] PELLERIN, D.,THIBAULT, S., 2005, *Practical FPGA Programming in C*. 1 ed. Indianapolis, USA, Prentice Hall PTR.

[35] PEDRETTI, K. T., CASAVANT, T. L., BRAUN, R. C., et al, 2001, "Parallelization of Local BLAST Service on Workstation Clusters", *Future Generation Computer Systems*, v. 17, pp. 745-754.

[36] COSTA, R. L. D. C., 2002, *Alocação de dados e Distribuição de Carga para Execução Paralela da Estratégia BLAST de Comparação de Sequencias*. Tese de M.Sc., Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil.

[37] COSTA, R. L. D. C.,LIFSCHITZ, S., 2004, "Workload Balancing on Genomic Databases for BLAST Processing", pp. 719-732.

[38] PACHECO, P. S., 1997, *Parallel Programming with MPI*. 1 ed. San Francisco, California, USA., Morgan Kaufmann Publishers, Inc.

[39] GROPP, W., LUSK, E., SKJELLUM, A., 1999, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. 2 ed. London, England, The MIT Press.

[40] BJORNSON, R. D., SHERMAN, A. H., WESTON, S. B., et al, 2002, "TurboBLAST: A Parallel Implementation of BLAST Built on the TurboHub".

[41] CHEN, R., TAAFFE-HEDGLIN, C., WILLARD, N., et al, 2002, "Benchmark and Performance Analysis of TurboBLAST on IBM® xSeries™ Server Cluster", *IBM Redbooks*.

[42] TONER, B., 2002, "BioInform Linux Cluster User Survey". Disponível em: http://bioinformatics.org/pipermail/bioclusters/2002-October/000432.html.

[43] GEIST, A., BEGUELIN, A., DONGARRA, J., et al, 1994, *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*. London, England, The MIT Press.

[44] COSTA, R. L. D. C.,LIFSCHITZ, S., 2003, "Database Allocation Strategies for Parallel BLAST Evaluation on Clusters", *Distributed and Parallel Databases*, v. 13, n. 1, pp. 99-127.

[45] PBS PRO(TM), 2004, *PBS Pro(TM) 5.4. User Guide*. 1 ed. Troy, Michigan, USA.

[46] MPICH. 2004, Disponível em: ftp://ftp.mcs.anl.gov/pub/mpi/mpich.tar.gz

[47] FENLASON, J.,STALLMAN, R., 1998, "GNU gprof: The GNU Profiler". Disponível em: www.gnu.org, http://www.gnu.org/software/binutils/manual/gprof-2.9.1/gprof.html.

[48] LANDMAN, J., 2004, "AMD Opteron (TM) Processor Benchmarking Results". Disponível em: http://scalableinformatics.com, http://enterprise2.amd.com/downloadables/Bioinformatics_performance.pdf.

[49] MEYER, L. A. V. C., RÖSSLE, S. C., BISCH, P. M., et al, 2004, "Parallelism in Bioinformatics Workflows", *VECPAR 2004: High Performance Computing for Computational Science*, v. 3402, n. Parallel and Distributed Computing, pp. 583-597.

[50] JOSUTTIS, N. M., 1999, *The C++ Standard Library: A Tutorial and Reference*. Vancouver, Canada, Addison Wesley.

[51] GNU. 2000, Disponível em: http://gcc.gnu.org/

[52] SDSC-UCLA. 1999, Disponível em: http://workbench.sdsc.edu/

[53] BLASTLAB, 2000, "How to BLAST Using Very Large Nucleotide Queries", *NCBI News*, n. Springer, pp. 7-7.National Institutes of Health.

[54] NEWHAM, C.,ROSENBLATT, B., 1998, *Learning the bash Shell.* Second ed. Sebastopol, California, USA, O'Reilly & Associates, Inc.

Apêndice A

Neste apêndice, é apresentado o levantamento estatístico das seqüências da base de dados completa do 'nt' para efeitos de comparação com o levantamento realizado para base de dados reduzida do 'nt' apresentado na seção 4.2.4 do capítulo 4.

Conforme pode ser observado no gráfico da Figura 65, o grupo um – seqüências com até 5.000 caracteres – engloba quase todas as seqüências, com cerca de 98% delas, porém conforme pode ser observado no gráfico da Figura 66, sua relevância em relação ao tamanho total da base de dados é bastante reduzida, caindo para 25%. O processo inverso ocorre com o quinto grupo – seqüências com mais de 20.000 caracteres – que sofre um salto de 1,9% para 70,85%.

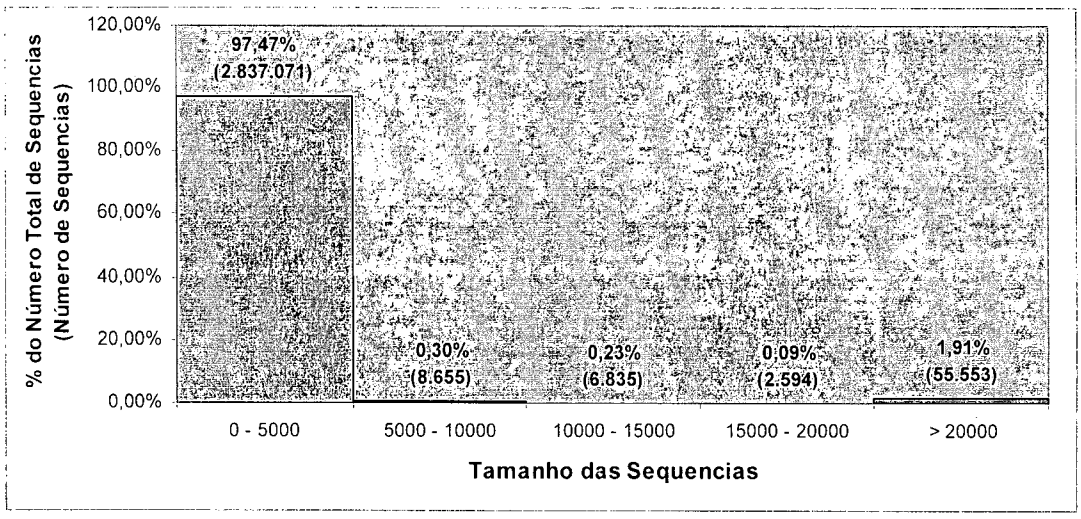

Figura 65 - Seqüências da Base de Dados Agrupadas por Tamanho

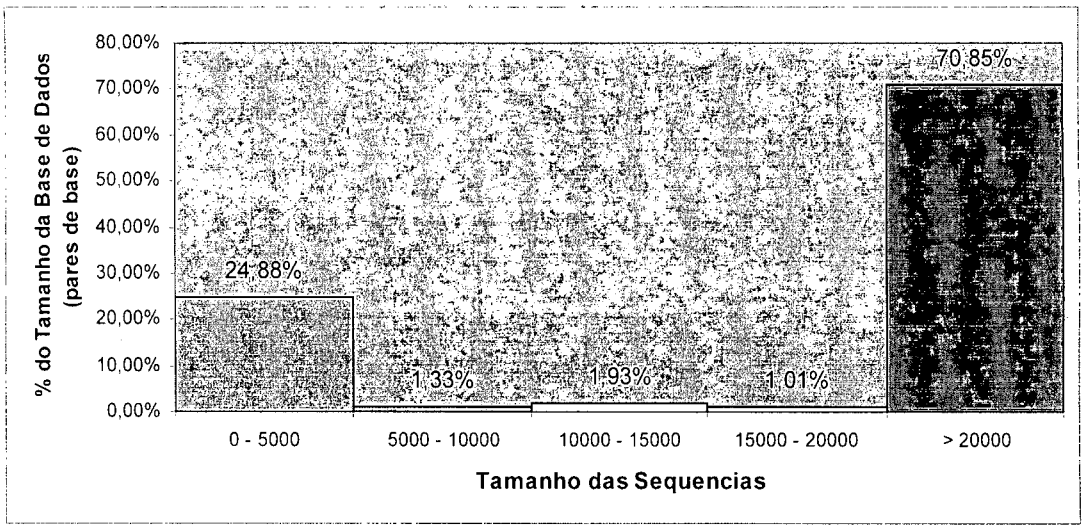

Figura 66 - Peso dos Grupos em Relação ao Tamanho Total da Base de Dados

Devido a grande quantidade de seqüências no grupo com ate 5.000 caracteres, um segundo levantamento foi feito para avaliá-lo em separado. Foram criados 20 grupos, de 50 caracteres cada, para agrupar as seqüências com até 1.000 caracteres, e mais um grupo, o 21, para as seqüências entre 1.000 e 5.000 caracteres.

Seguindo a tendência dos gráficos anteriores, as menores seqüências possuem um peso maior na quantidade de que no tamanho total. Os grupos de 1 a 6 – seqüências com até 300 caracteres – seguem esta tendência, como pode ser observado nos gráficos da Figura 67, Figura 68 e Tabela 17. As seqüências maiores por sua vez, apesar de em menor número possuem um peso maior em relação ao tamanho total, como é o caso do grupo 21 – seqüências com mais de 1000 caracteres – que sofre um salto de 7% para 29%.

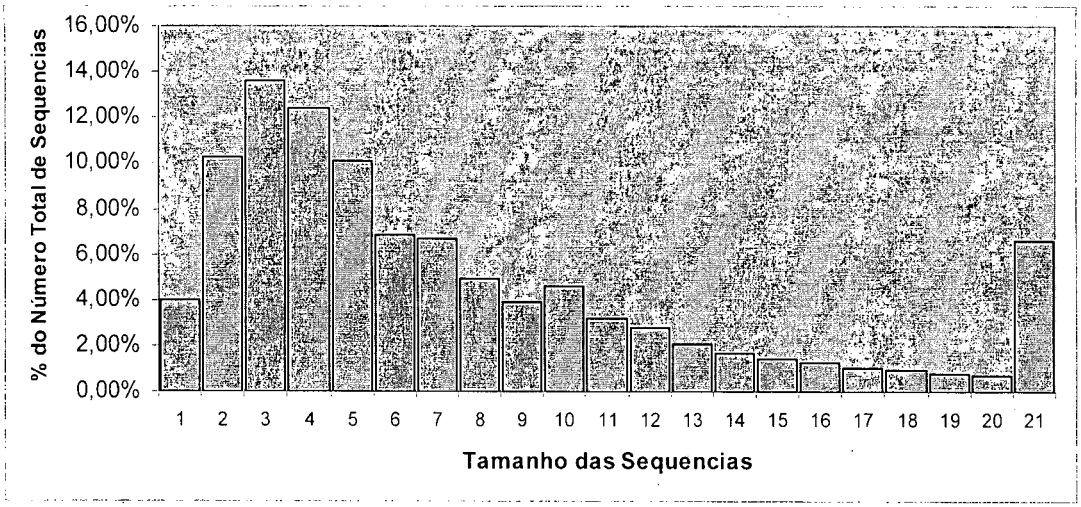

Figura 67 - Seqüências de ate 5.000 caracteres Agrupadas por Tamanho

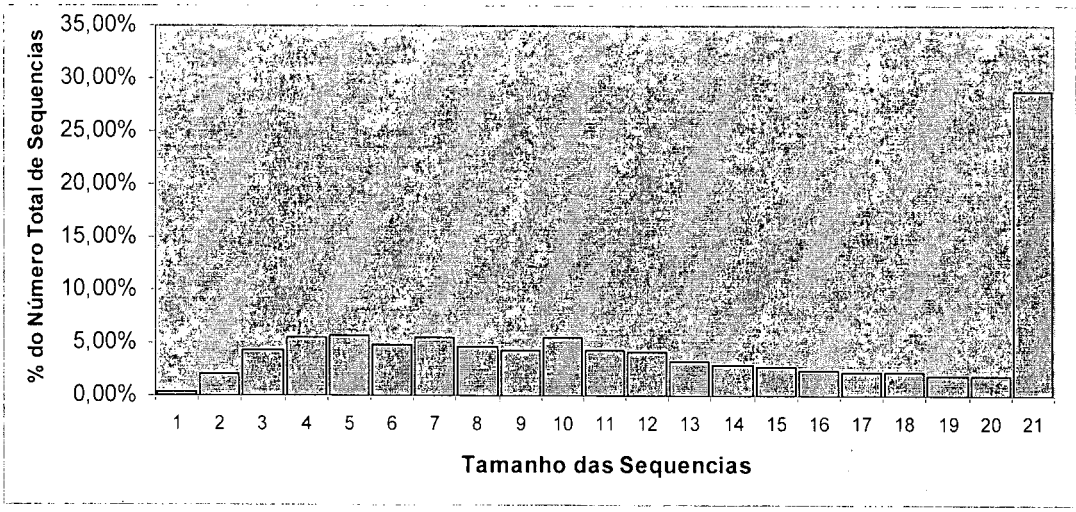

Figura 68 - Peso dos Grupos em Relação ao Tamanho Total

Tabela 17 - Percentual dos Grupos

	Faixa	% Num. Sequecias	% Tamanho		Faixa	% Num. Sequecias	% Tamanho
1	0 - 50	3,97%	0,31%	11	500 - 550	3,18%	4,27%
2	50 - 100	10,25%	2,07%	12	550 - 600	2,82%	4,16%
3	100 - 150	13,64%	4,38%	13	600 - 650	2,08%	3,33%
4	150 - 200	12,46%	5,59%	14	650 - 700	1,71%	2,95%
5	200 - 250	10,11%	5,77%	15	700 - 750	1,46%	2,70%
6	250 - 300	6,84%	4,82%	16	750 - 800	1,23%	2,45%
7	300 - 350	6,68%	5,56%	17	800 - 850	1,07%	2,25%
8	350 - 400	4,91%	4,71%	18	850 - 900	0,96%	2,16%
9	400 - 450	3,92%	4,27%	19	900 - 950	0,83%	1,98%
10	450 - 500	4,58%	5,59%	20	950 - 1000	0,74%	1,84%
				21	> 1000	6,57%	28,84%