Universidade de Caxias do Sul – UCS
Centro de Computação e Tecnologia da Informação - CCTI
Curso de Bacharelado em Ciência da Computação

Paralelização de Métodos de Bioinformática Utilizando Grids Computacionais

Mauricio Adami Mariani

Prof. André Luis Martinotto
Orientador

Prof. Günther Johannes Lewczuk Gerhardt
e
Scheila de Avila e Silva
Co-orientadores

Caxias do sul, dezembro de 2009

Quero legar a meu filho

Quero legar a meu filho
além da doçura
a coleção de blasfêmias
e palavrões que conheço;
para que ele saiba
o que é ofender e o que é ofendido;

deixar poucos do metal
que se acumula,
mas dar-lhe um bilhão
de livros, poemas, e ciências,
para que ele saiba
do vermelho e do branco.

E dar-lhe sons e cores
fazê-lo ver o caos e o sexo
de todas mulheres imagináveis
e que ele saiba enfim,
que o amor é chama
e tanto exige.

Valha-me sagração de tudo
se não mostrarei a ele
que a vaidade corrompe
a injustiça violenta,
para que saiba chorando
o que fizeram em dois mil anos.

Quero legar a meu filho
que a pedra pedra,
que a água água, e a flor
é flor, nada além disso,
é bom que ele descubra
o justo, a certeza, o constatar.

- Assis Antônio Mariani

AGRADECIMENTOS

Inicialmente gostaria de agradecer a minha mãe, Miriam Mariani, que me proporcionou condições para todo meus estudos. Também gostaria de agradece-la por toda sua força e paciência na vida.

Gostaria de agradecer a minha namorada, Patrícia Marin, por todo apoio e conforto que me deu nas piores horas. E também pela sua paciência e carinho.

Gostaria de agradecer ao meu orientador pelo apoio e infinitas correções da monografia, sem o qual não teria terminado esse trabalho.

À minha co-orientadora, Scheila, por suas correções, amizade e apoio.

Ao meu co-orientador, Günther, responsável pelo tema da monografia e apoio.

ÍNDICE

RESUMO

Os recentes avanços da computação beneficiaram diversas áreas de pesquisa. Entre essas uma área com grande destaque é a biologia, que é o campo de estudo desse trabalho. Os avanços tecnológicos permitiram um aumento na capacidade dos processadores e, consequentemente, permitiram que uma quantidade maior de dados fosse analisada, e métodos mais complexos fossem utilizados.

O uso de métodos nas pesquisas em bioinformática vem crescendo. Porém, quando na análise de sequências de DNA de organismos mais complexos, que possuem milhares de pares de bases, necessita-se de uma maior capacidade de processamento do que a disponibilizada por um único computador. Devido ao aumento da quantidade de dados a serem processados e com a necessidade de resultados mais imediatos, o uso de arquiteturas computacionais de alto desempenho, bem como, de programas específicos para essas arquiteturas são necessários.

Dentro desse contexto nesse trabalho é feito um estudo do uso de computação de alto desempenho no processamento de sequências de DNA. Mais especificamente na paralelização de métodos utilizados pelo grupo de pesquisas em bioinformática da UCS. Para o desenvolvimento desse trabalho optou-se pela utilização de *grids* computacionais, uma vez que esse tipo de plataforma disponibiliza uma alta capacidade de processamento a um baixo custo.

Palavras-chaves: bioinformática, DNA, *grid* computacional.

ABSTRACT

Recent advances in computing received several research areas. These an area with great emphasis is biology, which is the field of study of this work. Advances technology enabled an increase in the capacity of processors and therefore allowed a larger amount of data was analyzed, and more complex methods were used.

The use of methods in research in bioinformatics has grown. However, when the analysis of DNA sequences from more complex organisms, which have thousands of pairs of bases, it requires greater processing power than that given by a single computer. Due to the increasing amount of data to be processed and the need for more immediate results, the use of computing architectures for high performance, as well as specific programs for these architectures are needed.

Within this context this work, we study the use of computing capabilities performance in processing DNA sequences. More specifically the parallelization of methods used by the research group in bioinformatics at UCS. For development this work we chose to use computational grids, since this type of platform provides a high processing capacity at low cost.

Keywords: bioinformatics, DNA, computational grids.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

A	Adenina
AM	Aprendizado de Máquina
BOINC	*Berkeley Open Infrastucture For Network Computing*
C	Citosina
DARPA	*Defense Advanced Research Projects Agency*
DNA	*Desoxyribonucleic acid*
FC	Função de Correlação
FTP	*File Transfer Protocol*
G	Guanina
HP	*Hewlett-Packard*
IBM	International Business Machines
IP	*Internet Protocol*
MIMD	*Multiple Instruction Multiple Data*
MPI	*Message Passing Interface*
mRNA	RNA mensageiro
NASA	National Aeronautics and Space Administration
NUMA	*non-uniform memory access*
OpenMP	*Open Multi-Processing*
P2P	*Peer-to-Peer*
PC	*Personal Computer*
PVM	*Parallel Virtual Machine*
RN	Rede Neural
RNA	*Ribonucleic acid*
T	Timina
TCP	*Transmission Control Protocol*
U	Uracila
UCS	Universidade de Caxias do Sul
UFCG	Universidade Federal de Campina Grande
UMA	*uniform memory access*

1 INTRODUÇÃO

O uso da computação vem crescendo em diversas áreas da ciência, como por exemplo, matemática, física, química, engenharia, entre outras. Uma área que apresenta uma necessidade computacional cada vez maior é a biologia, que é o objeto de estudo deste trabalho (LESK, 2008).

A biologia é o ramo da ciência que estuda as características e o comportamento dos organismos, a origem de espécies e indivíduos, bem como a forma como esses interagem uns com os outros e com o seu ambiente. Essas características e comportamentos são definidos pelo *DNA* (*Desoxyribonucleic acid*). Assim, o estudo do *DNA* pode prover informações sobre o genótipo (dados genéticos de um organismo) (LESK, 2008).

Inicialmente o estudo e o processamento do *DNA* era realizado de forma manual, e posteriormente passou-se a utilizar computadores nessa tarefa. Porém com o aumento da quantidade de dados, junto com a necessidade de resultados mais imediatos, deu-se a necessidade do uso de arquiteturas computacionais de alto desempenho, bem como, de programas específicos para essas arquiteturas.

Como uma primeira alternativa de um ambiente de alto desempenho tem-se os supercomputadores que podem ser caracterizados, principalmente, pela tecnologia avançada empregada no seu desenvolvimento e pela alta capacidade de processamento (JACOB ET AL., 2005). Geralmente, essas máquinas ocupam grandes salas, ou até andares de um prédio e são formados por vários *racks* (unidades com processadores e memória) ligados, através de uma rede proprietária e local, de forma a trabalharem juntos na resolução de um problema específico. Esses equipamentos se caracterizam por serem formados por tecnologias avançadas e frequentemente proprietárias e, em alguns casos, são desenvolvidos especialmente para grandes instituição ou organização que estão adquirindo-o. Devido ao alto custo desse tipo de arquitetura sua aquisição é restrita, basicamente a órgãos de pesquisa, instituições governamentais e grandes corporações (JACOB ET AL., 2005).

Uma outra opção para obtenção de um alto poder computacional é o uso de c*lusters* de computadores, que consistem em um ambiente de computação paralela formado por um conjunto de computadores pessoais (PC's), interligados por uma rede local de alto desempenho. Apesar de terem um custo, na sua totalidade, bem inferior a de um supercomputador esse tipo de arquitetura ainda apresenta um custo relativamente alto, devido a necessidade de investimentos na compra dos computadores e de equipamentos de rede adequados, bem como, a necessidade de manutenção e atualização desses equipamentos (BUYYA, 1999).

Como uma alternativa ao uso de supercomputadores e c*lusters* surgiram os g*rids* computacionais. Esses baseiam-se na utilização de computadores não dedicados para o desenvolvimento de uma plataforma de alto desempenho. Os *grids* computacionais visam a utilização de ciclos ociosos de computadores para a solução de problemas que necessitam de um alto poder de processamento. Uma vez que diversas instituições, como por exemplo a Universidade de Caxias do Sul, possuem um grande número de computadores que permanecem ociosos em diversos períodos

do dia essa plataforma torna-se atrativa, pois permite a obtenção de um alto poder computacional através de um baixo investimento em equipamentos (FOSTER, 1995).

Dentro desse escopo, esse trabalho tem como principal objetivo a utilização de *grids* computacionais no processamento de sequências de *DNA*. A partir da análise dessas sequências pode-se entender melhor a estrutura de uma determinada sequência de *DNA*, bem como compreender a função da mesma. Atualmente existem muitas ferramentas que implementam diversos métodos para a análise de *DNA,* dentre esses métodos pode-se destacar os utilizados pelo grupo de pesquisa do Prof. Dr. Günther Johannes Lewczuk Gerhardt, que são: análise de organização do DNA, análise da periodicidade do DNA, cálculo do coeficiente de clusterização de uma sequência de DNA, cálculo da dispersão do coeficiente de clusterização e uso da técnica de aprendizado de máquina para determinar se uma sequência é promotora.

Frequentemente os dados utilizados como entrada (sequências de *DNA*) para esses métodos são obtidos de repositórios *online* sem acesso restrito, como por exemplo o GenBank (WHEELER et al, 2006). Esses repositórios foram criados com o objetivo de armazenar em um único lugar e disponibilizar em um formato comum as sequências de *DNA* já pesquisadas. Essas bases de dados são utilizadas, sejam na obtenção dos dados ou em atualizações, por centenas de instituições, e por isso possuem milhares de sequências de *DNA*, e continuam a crescer constantemente. Assim, devido à grande quantidade de dados disponíveis nesses repositórios e o constante crescimento dos mesmos, o processamento desses dados ou um subconjunto deles é inviável de ser realizado em uma único computador, tornando-se necessário o uso de computação de alto desempenho.

Nesse trabalho serão analisados os métodos e ferramentas para o processamento de DNA utilizados pelo grupo de pesquisas em bioinformática da UCS, a fim de adaptá-los para o uso em g*rids*. Para testes das implementações desenvolvidas serão utilizados dados obtidos através do repositório GenBank.

1.1 Objetivos

Este trabalho teve como objetivo paralelizar alguns métodos, que desempenham funções de análise básica em seqüências de *DNA* na área da biologia molecular. Entre esses métodos foram estudados : análise de organização do DNA, análise da periodicidade do DNA, cálculo do coeficiente de clusterização de uma sequência de DNA, cálculo da dispersão do coeficiente de clusterização e uma técnica de aprendizado de máquina para determinar se uma sequência é promotora.

Após o estudo inicial desses métodos foi realizada uma análise para a adaptação desses para uma plataforma de *grid* computacional. Esse estudo visou a execução desses métodos em um ambiente distribuído, com o objetivo de reduzir o tempo de execução dessas tarefas. Optou-se pelo uso de *grids* computacionais devido ao fato desses fornecerem um ambiente de alto desempenho com baixo investimento.

1.2 Organização do trabalho

O presente trabalho está estruturado da seguinte forma:

- Capítulo 1: foi feita uma breve introdução sobre o trabalho desenvolvido. Nesse foram apresentados ainda os objetivos do trabalho.

- Capítulo 2: são abordados os conceitos de biologia molecular necessários para a compreensão das pesquisas realizadas pelo grupo de bioinformática da UCS. Neste capítulo também foram descritos os métodos utilizados pelo grupo de bioinformática da UCS, que foram os métodos adaptados para a execução em *grids* computacionais.

- Capítulo 3: nesse foram introduzidos os principais conceitos sobre computação paralela. Foram vistos ainda conceitos sobre *grids* computacionais, que foi a plataforma usada no desenvolvimento desse trabalho. Por fim, foram descritas as principais ferramentas utilizadas para o gerenciamento desse tipo de arquitetura.

- Capítulo 4: nesse tem-se uma descrição das implementações desenvolvidas no trabalho. Além disso, são apresentados os resultados obtidos a partir da execução dessas implementações em um ambiente de *grid*.

- Capítulo 5: nesse são descritas as conclusões do trabalho juntamente com sugestões de trabalhos futuros.

2 BIOLOGIA MOLECULAR

Este capítulo tem como objetivo descrever o dogma central da biologia. Nesse serão abordados os conceitos básicos e relevantes sobre este tema a fim de tornar compreensível os objetivos e a importância dos métodos computacionais empregados na área da bioinformática.

A bioinformática consiste na aplicação de técnicas computacionais no campo da biologia, e compreende em um vasto campo para desenvolvimentos e pesquisas, como por exemplo, a criação de bases de dados, o uso de algoritmos computacionais e técnicas estatísticas para resolver problemas práticos decorrentes da análise de dados biológicos (GIBAS;JAMBECK, 2001).

Nas últimas décadas a evolução nas técnicas de genômica, das tecnologias de pesquisa e o desenvolvimento da computação têm-se combinado de forma a produzir uma quantidade enorme de informações relacionadas à biologia molecular e aumentar a compreensão dos processos biológicos (LESK, 2008). Entre os métodos computacionais mais comuns usados no processamento de informação biológica pode-se citar: mapeamento, alinhamento, comparação e análise de sequências de DNA e proteínas, assim como a visualização de modelos em três dimensões de estruturas de proteínas e mineração de dados (GIBAS;JAMBECK, 2001).

2.1 Molécula de DNA

As características e a continuidade dos organismos dependem do material genético presente nas células. O DNA é uma molécula constituída por duas fitas formadas por unidades repetitivas, denominadas nucleotídeos, ligadas entre si, como pode ser visualizado na Figura 1. Existem quatro tipos de nucleotídeos que são denotados pelas letras A (Adenina), T (Timina), C (Citosina), G (Guanina) como ilustrado na Figura 1 (LEWIN, 2001).

Esses quatro nucleotídeos são formados por três subunidades: um grupo de fosfato; uma pentose (açúcar com cinco átomos de carbono) e um composto de bases nitrogenadas, que podem ser púricas possuindo dois anéis com átomos de nitrogênio, ou pirimídicas que contém somente um anel com átomos de hidrogênio, conforme pode ser observado na Figura 1 (LEWIN, 2001).

Em uma molécula de DNA as duas fitas de nucleotídeos se ligam de forma que o nucleotídeo A de uma fita se liga com o nucleotídeo T da outra fita através de duas pontes de hidrogênio e o nucleotídeo C se liga com o nucleotídeo G através de três pontes de hidrogênio (conexões entre as bases nitrogenadas) como pode ser visualizado na Figura 1 (LEWIN, 2001).

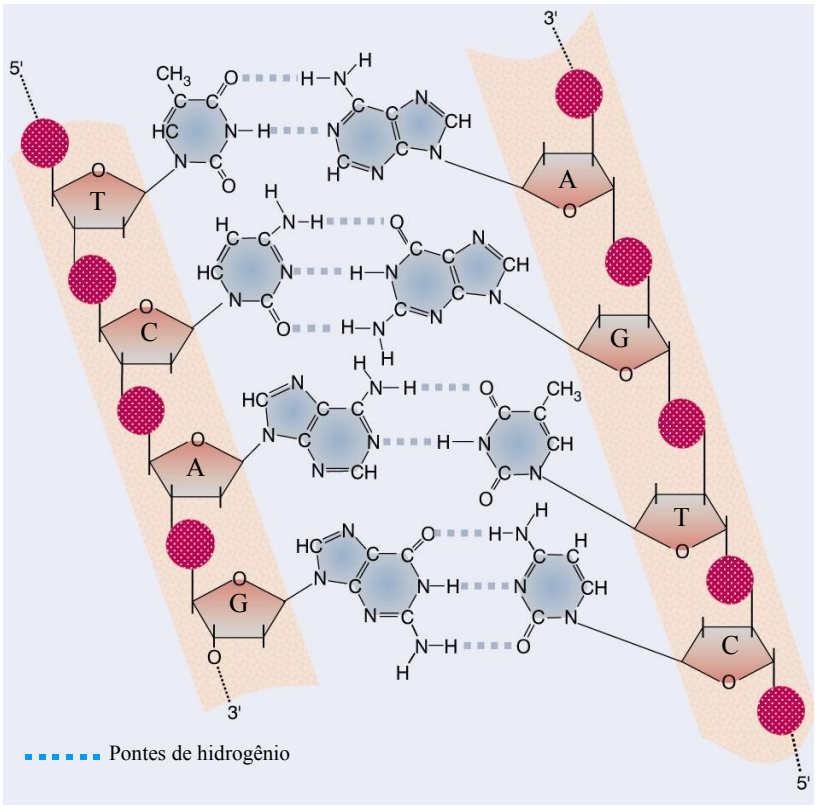

Figura 1: Ligações entre os nucleotídios A-T com duas pontes de hidrogênios e G-C com três pontes de hidrogênios. A sequência é T-A, C-G, A-T, G-C (LEWIN, 2001).

2.2 Dogma Central da biologia

O dogma central define o paradigma da biologia molecular e compreende três processos: a replicação, a transcrição e a tradução do DNA. A replicação do DNA é um processo que ocorre quando uma célula se divide para formar outra. Esse processo é responsável pela passagem da informação de uma geração a outra. O processo de transcrição converte a informação do DNA em uma fita de RNA mensageiro (mRNA), que servirá para a tradução. Já o processo da tradução converte a informação contida no mRNA em proteínas (LEWIN, 2001).

A duplicação ou replicação acontece quando as fitas do DNA se separam e servem como molde para a síntese de novas fitas de DNA, formando novamente uma dupla hélice idêntica à original. Esse tipo de replicação é do tipo semi-conservativa, pois cada dupla hélice gerada contém um filamento antigo e outro recém-sintetizado (NELSON;COX,2002).

A transcrição é o processo no qual uma das cadeias de DNA, denominada fita-molde, forma uma cadeia única de mRNA, substituindo os nucleotídios A por U (Uracila), T por A, G por C e C por G, como pode ser visualizado na Figura 2 (LEWIN, 2001). O mRNA leva a informação transcrita a partir da molécula de DNA do núcleo para o citoplasma, onde ocorre a tradução. Cada três nucleotídeos no

mRNA são denominados códon e correspondem a um aminoácido na proteína que irá se formar (VOET, 2006).

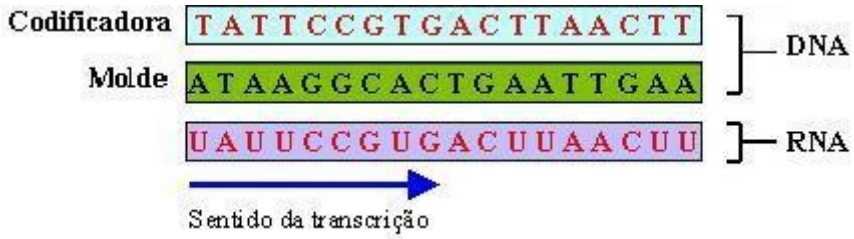

Figura 2: Transcrição da fita-molde do DNA para RNA

A tradução é o processo que ocorre nos ribossomos, onde o mRNA é lido e traduzido em uma biomolécula específica, que pode ser uma proteína, uma outra molécula de RNA, etc. Como a molécula de mRNA se desloca pausadamente no ribossomo, ocorre a leitura da seqüência de três nucleotídeos que são traduzidos em um aminoácido, isso ocorre até o final da sequência, onde existe um ponto de parada. Assim, a sequência de mRNA transforma-se em uma sequência correspondente de aminoácidos, para produzir uma proteína (LEWIN, 2001). Ao fim desse processo tem-se a expressão gênica que é o gene funcional resultante da transcrição e tradução.

Os tipos e a sequência de aminoácidos (unidade fundamental das proteínas) na cadeia polipeptídica são determinados pela sequência de bases do mRNA. Como este é fabricado pela transcrição rigorosa de uma das cadeias de DNA, conclui-se que a sequência e os tipos de aminoácidos em uma proteína são determinados pela sequência de bases do DNA, ou seja, pelo gene.

2.3 Ferramentas utilizadas pelo grupo de bioinformática da UCS

Na UCS existe o grupo de pesquisa em bioinformática com vários trabalhos sendo desenvolvidos nessa área (GERHARDT;PANIS;SANTOS,200X) (GERHARDT;LEMKE;CORSO,2005) (SILVA,2006)(SANTOS,2009)(GERHARDT *et.* al.,2004), sendo que em cada um deles utiliza-se de ferramentas e métodos adequados aos seus objetivos. Aqui serão descritos brevemente os métodos usados nas pesquisas dirigidas pelo Prof. Dr. Günther J. L. Gerhardt.

2.3.1 Cálculo da Entropia

A entropia é uma grandeza termodinâmica geralmente associada ao grau de desordem de um sistema e essa mede a parte da energia que não pode ser transformada em trabalho. Essa, é também a medida natural para os seguintes fenômenos: complexidade, compressibilidade, previsibilidade e aleatoriedade (FARACH *et* al, 1994). Um dos principais motivos para o cálculo da entropia ser utilizado em moléculas de DNA está no fato de que com a sua utilização pode-se obter o grau de organização

de uma estrutura. Na Figura 3 tem-se uma sequência de DNA totalmente organizada, ou seja, uma sequência cuja entropia é 0.

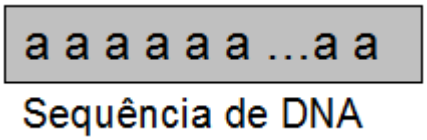

Figura 3: Sequência totalmente organizada, com entropia 0, pois existem somente nucleotídios a.

Quando executa-se o cálculo da entropia em uma mensagem, por exemplo, provavelmente será conhecida a estrutura por trás dessa mensagem. O motivo de aplicar o método da entropia em seqüências de DNA está na busca de padrões (HORTA, 2001). Através destes padrões é possível buscar regiões do DNA que possuam significado próprio, como os limites das áreas codificadoras do DNA, ou mesmo buscar algum padrão e, a partir, deste verificar a existência de algum significado biológico (FARACH et. al., 1994). A aplicação do método de entropia também visa a busca de relacionamentos entre sequências distintas de DNA.

Para o cálculo da entropia, utiliza-se como entrada uma sequência de DNA denominada janela. Essa sequência *S* é composta por *k* nucleotídios onde cada elemento $x_i \in \{a, t, c, g\}$ e pode ser escrita como $S=\{x_1, x_2, \ldots, x_k\}$. Inicialmente esse método considera a quantidade de tripletes iguais existentes na sequência de DNA, começando em x_1, x_2, x_3 e deslocando-se uma posição para direita, assim o segundo triplete será x_2, x_3, x_4 e assim por diante. Desta forma obtém-se um conjunto $D(S)=\{j_1, j_2, \ldots, j_n\}$, com *n* posições, formado por $j_1=Q(x_1 x_2 x_3), j_2=Q(x_2 x_3 x_4), \ldots, j_k=Q(x_{k-2} x_{k-1} x_k)$, onde *Q()* representa o número de vezes em que cada sequência (única) de tripletes aparece na janela. Este conjunto pode ter no máximo 64 posições, ou seja, 4 possibilidades de nucleotídios combinados em três posições.

Para o conjunto *D(S)*, pode-se construir uma distribuição de probabilidade discreta $P=(p_1, p_2, \ldots, p_n)$, sendo p_i a probabilidade de ocorrência de cada sequência de tripletes e onde $\sum_i p_i = 1$, resultando em 1 pois é a soma das probabilidades p_i de cada triplete é $\left(p_i=\frac{j_i}{k}\right)$. Após obter-se a distribuição *P* pode-se calcular a entropia de Shannon de uma distribuição de probabilidade discreta através da fórmula (SHANNON, WEAVER, 1963):

$$H(S)=H(D(S), P)=-\sum_{i=1}^{n} p_i \log_2 p_i$$

A implementação sequencial do método de entropia pode ser vista na Seção A.1 do Anexo A. Essa foi desenvolvida pelo Prof. Dr. Günther Johannes Lewczuk Gerhardt utilizando a linguagem C.

2.3.2 Busca de Periodicidades

Este método procura estabelecer relações biológicas entre os vários padrões de repetição existentes ao longo de uma sequência. A busca de periodicidades ocorre por meio do uso de funções de correlação (FC), que realizam a contagem das regiões de nucleotídeos em um determinado intervalo. Por exemplo, se a periodicidade for 3, será feita uma comparação de um nucleotídeo com seus dois vizinhos subjacentes, como visualizado na Figura 4.

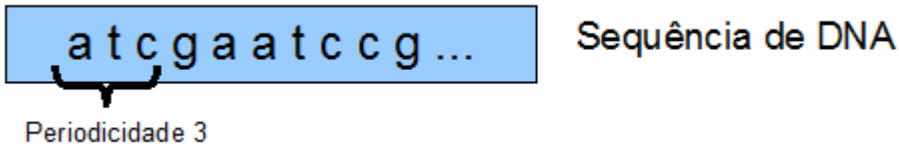

Figura 4: Distância de 2 nucleotídios caracteriza a periodicidade 3

Sobre essa FC, podem ser utilizadas várias ferramentas que mapeiam ou localizam essas regiões, como por exemplo, a transformada de Fourier, que mapeia a FC em um espaço de freqüências. A periodicidade 3 é a mais estudada já que é indicativa de regiões codificadoras (GERHARDT *et.* al., 2004).

O método de busca de periodicidades utiliza-se de uma janela inicial de tamanho *k*, onde é aplicada a função de correlação de Kronecker para a comparação entre cada caractere da janela estudada. O algoritmo trata a sequência de nucleotídios como uma cadeia circular (GERHARDT, SILVA, PINTO, 2004).

Na função de correlação de Kronecker cada nucleotídio é comparado com todos os demais da janela. A comparação entre dois elementos retorna 1 caso forem iguais e 0 caso forem diferentes. Assim, *S(1)* é a soma resultante da comparação entre os elementos de distância 1, *S(2)* é a soma da comparação entre os elementos com distância 2 e assim sucessivamente. Então *S(k)* é o valor da função de correlação para uma distância *k* entre elementos da seqüência. Essa função é representada matematicamente por

$$S(k)=\frac{\sum_{i=F}^{L}\delta_{i,((i+k+F)\,mod\,(L-F+1))+F}}{(L-F+1)}$$

onde *F* e *L* representam o primeiro e último elementos da janela, respectivamente. O operador *mod* retorna o resto da divisão entre os termos *i+k+F* e *L-F+1*, e Delta é a função delta de Kronecker definida por:

$$\delta_{i,j}=\begin{cases}0\ para\ x\neq y\\1\ para\ x=y\end{cases}$$

onde *i* e *j* representam as posições dos elementos e *x* e *y* representam as bases contidas nessas posições. Assim, nessa função de correlação se as bases contidas em *i* e *j* forem iguais, a função *Delta* de Kronecker retorna 1, e caso contrário, retorna 0. Com o uso dessa função de correlação pode-se obter a relação existente entre cada elemento da janela em relação a todos os outros elementos da mesma janela.

Em seguida, é retirado o DC (*Direct Current*) do resultado da função de correlação. Isso é feito subtraindo-se de cada resultado a média dos mesmos, para tanto, e necessário somar todos os resultados da função de correlação e dividir pelo tamanho da janela escolhida. Esse processo também é conhecido por *Zero Frequency*, pois alinha o resultado da função de correlação em relação a *y = 0*, facilitando a análise dos dados.

Sobre o resultado da remoção de DC é aplicada a Transformada de Fourier. A partir dessa é possível visualizar a correlação de cada nucleotídio por meio de um gráfico de frequência. Essa transformação dos valores em espaços de frequência é dada pela equação:

$$F\{S(n)\}(w) = \int_{-\infty}^{\infty} \frac{S(n)\mathrm{e}^{-2\Pi win}\,dn}{\sqrt{2\Pi}}$$

onde *i* é a unidade imaginária $(\sqrt{-1})$ e *e* o número de Euler. Assim com a aplicação de transformada de Fourier cria-se um espaço de frequências de *S* em função do deslocamento *k*, onde *F(w)* será o sinal dado por *S(k)* no domínio da frequência, ou seja, seu espectro.

A implementação do método de cálculo de periodicidade pode ser vista na Seção A.2 do Anexo A. Essa implementação foi desenvolvida pelo Prof. Dr. Günther Johannes Lewczuk Gerhardt utilizando a linguagem C.

2.3.3 Cálculo do coeficiente de clusterização

Utilizar a teoria dos grafos para analisar o coeficiente de clusterização de uma sequência é outro método utilizado neste trabalho. Inicialmente a sequência de DNA é transformada em um grafo (GERHARDT;LEMKE;CORSO,2005), onde cada vértice do grafo corresponde a um triplete de nucleotídeos, sendo que dois vértices estão ligados por uma aresta se seus nucleotídeos são justapostos (SANTOS,2009), como pode ser visualizado na Figura 5.

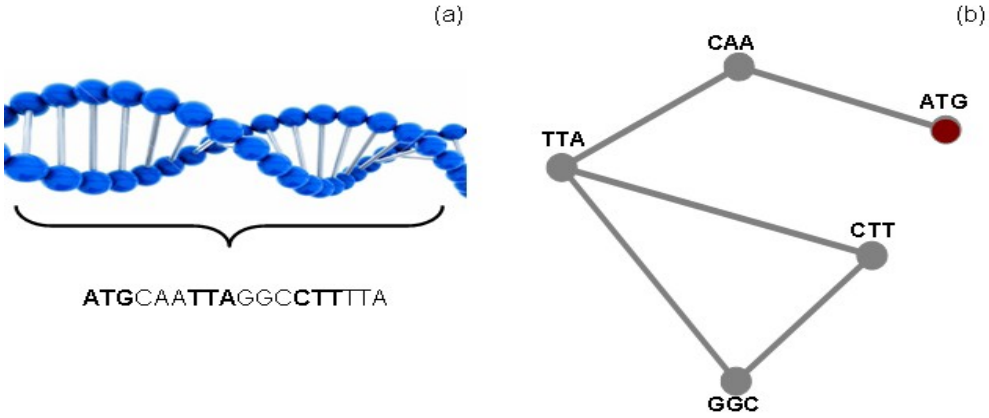

Figura 5: (a) Sequência de nucleotídios. (b) Grafo da sequência (a). Cada vértice é um triplete de nucleotídios começando em ATG.

Após, no grafo resultante é utilizado o método de clusterização que verifica a densidade de triângulos, a fim de determinar se esse grafo é organizado. Essa medida permite verificar se os grafos possuem ou não um padrão em sua estrutura. Se o padrão não existe, seu coeficientes de clusterização é semelhante ao de um grafo aleatório; porém, quando se tem um padrão na estrutura, pode-se achar características evolucionárias (GERHARDT;LEMKE;CORSO,2005).

Esse método possui como entrada uma sequência de nucleotídios *S*. Inicialmente esse método transforma essa sequência *S* em um grafo não direcionado, onde os vértices são tripletes de nucleotídios e uma ligação entre dois tripletes está estabelecida, quando esses são justapostos na sequência de DNA. O método utiliza os dois primeiros tripletes que são justapostos e atribui a cada triplete um único valor, formando uma aresta entre esse dois vértices (valores), e assim sucessivamente pelos tripletes justapostos até o final da sequência *S*. Para a representação do grafo utiliza-se uma estrutura de matriz de adjacência *m*, de ordem *N*, onde o elemento $m_{i,j}$ é 1 se o vértice *i* é conectado ao o vértice *j* e 0 de se não forem conectados.

O método de clusterização tem como objetivo calcular o nível de agrupamento dos vértices. Esse nível de agrupamento é dado por um valor c_i, que representa o coeficiente de clusterização para um vértice *i*, isto é, calcula-se o número de arestas existentes entre os vértices adjacentes ao vértice *i*. Se o vértice *i* é conectado ao um subgrafo *C*, o número de ligações entre os vértices desse subgrafo *C* é calculado por:

$$\mathscr{L}_i = \sum_{j=1}^{L} m_{i,j} \left[\sum_{k \in C} m_{i,k} \right]$$

Para encontrar c_i deve-se normalizar $\mathscr{L}_i$ para todas ligações possíveis k_i entre do vértice *i* e os vértices do subgrafo *C*, através de:

$$c_i = \frac{2\mathscr{L}_i}{k_i(k_i - 1)}$$

Quando $k_i = 0$ ou $k_i = 1$, pode-se definir $c_i = 0$. Finalmente *C*, que corresponde ao coeficiente de clusterização para o grafo, é obtido através da média dos coeficientes de clusterização de todos os subgrafos, como segue:

$$C = \frac{1}{L} \sum_{i=1}^{L} c_i$$

A implementação sequencial do método de cálculo do coeficiente de clusterização pode ser vista na Seção A.3 do Anexo A. Essa foi desenvolvida pelo Prof. Dr. Günther Johannes Lewczuk Gerhardt utilizando a linguagem C.

2.3.4 Cálculo da dispersão do coeficiente de clusterização

Pode-se aplicar também o cálculo da dispersão sobre um coeficiente de clusterização. O cálculo de dispersão é uma medida estatística utilizada para determinar a localização de uma sequência de DNA em relação a outra gerada como grupo de controle (SANTOS, 2009). Inicialmente deve ter-se a rede de controle, que, frequentemente, consiste em um grafo aleatório. Essa rede de controle varia seu conteúdo GC ao longo da sequência até conter somente nucleotídeos G e C (SANTOS, 2009). Aplicando-se o cálculo da dispersão, obtém-se o quanto um grafo de uma sequência de DNA real se assemelha a um grafo aleatório. Na Figura 6 tem-se um gráfico que ilustra esse método. No gráfico, a linha contínua representa a sequência aleatória e os pontos representam a sequência real de DNA. O cálculo do coeficiente de dispersão pode ser importante para estabelecer a evolução no DNA de um organismo (SANTOS, 2009).

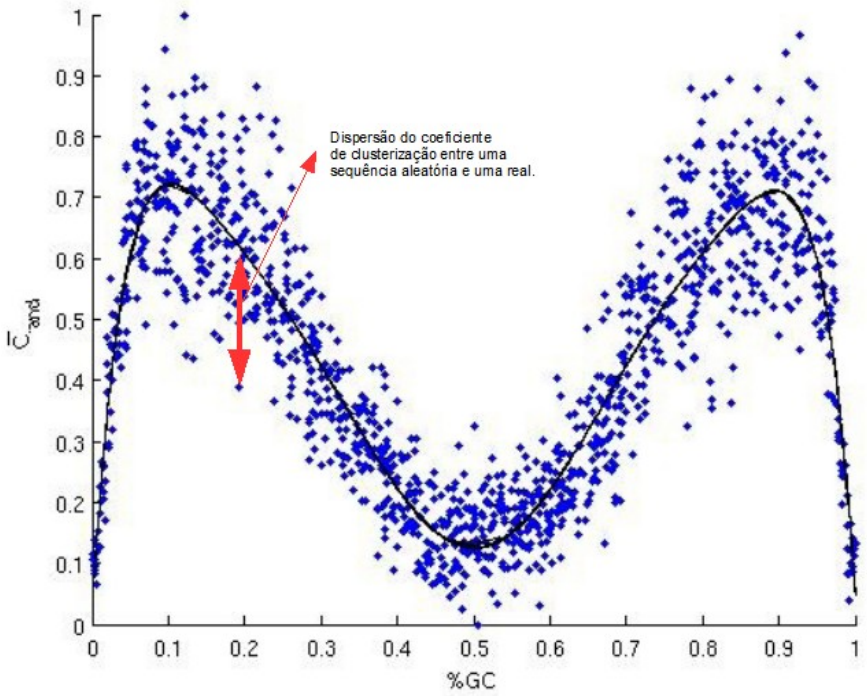

Figura 6: A curva preta representa uma sequência de DNA aleatória, e os pontos pertencem a sequências de DNA reais.

O método para o cálculo da dispersão possui como parâmetro de entrada os coeficientes de clusterização calculados pelo método descrito na Seção 3.1.3. A partir dessas informações o método calcula o coeficiente de dispersão *D* através da soma da diferença, entre o coeficiente de clusterização de um grafo real e do coeficiente de clusterização de um grafo randômico, dividida pelo desvio padrão. Essa soma será o coeficiente de dispersão do grafo, ou seja, da sequência de nucleotídios em relação a uma sequência de controle. O coeficiente de dispersão *D*, usado nesse trabalho, é definido como (GERHARDT *et* al., 2006):

$$D(C)_L = \frac{1}{n_b} \sum_{i=1}^{n_b} \frac{C_i - C_{rand}}{\sigma_{rand}}$$

onde n_b é o número de janelas de tamanho L da sequência analisada e C_{rand} é o coeficiente de aglomeração médio do grupo de controle e C_i, é o coeficiente de clusterização do grafo. Nesse contexto, σ representa o desvio padrão de C_{rand}, que pode ser encontrado numericamente.

A implementação sequencial do método de cálculo da dispersão do coeficiente de clusterização foi desenvolvida pelo Prof. Dr. Günther Johannes Lewczuk Gerhardt utilizando a linguagem C e pode ser vista na Seção A.4 do Anexo A.

2.3.5 Determinação de promotores através de uma rede neural

Para que um determinado gene tenha sua expressão gênica realizada de maneira correta, uma sequência anterior a ele deve ser reconhecida. Esta sequência é denominada de promotor, já que promove ou inibe a transcrição de um gene subseqüente a ele (LEWIN,2001). Reconhecer e predizer essas sequências, difíceis de serem identificadas experimentalmente, implica em conhecer a rede de regulação metabólica de um determinado organismo. Muitas técnicas de aprendizado de máquina (AM) são empregadas para reconhecer os promotores, dentre elas destacam-se, as redes neurais artificiais (SILVA, 2007).

Uma Rede Neural (RN) é um sistema de AM inspirado no funcionamento de redes neurais biológicas. A RN é considerada um processador paralelamente distribuído constituído de unidades de processamento simples interconectadas ou não. Essas unidades são chamadas de neurônios e tem a propensão natural de armazenar conhecimento experimental e torná-lo disponível para o uso (HAYKIN, 2005).

Pode-se afirmar que as RNs aprendem a partir dos exemplos. Uma RN é caracterizada: pelo padrão de conexões entre os neurônios (chamado de arquitetura); pelo método de determinação de pesos nas conexões (chamado de treinamento); e pela sua função de ativação (MOUNT, 2000). No grupo de bioinformática da UCS utiliza-se a RN para a predição e o reconhecimento de sequências promotoras em bactérias.

Os neurônios (ver Figura 7) são conectados por vínculos orientados, sendo que cada vínculo contém um peso numérico W_{ji} associado a ele, o qual determina a intensidade e o sinal da conexão. Além disso cada vínculo possui uma ativação a_j (RUSSELL;NORVIG,2004). A função da entrada é dada por:

$$en_i = \sum_{j=0}^{n} W_{ji} a_j$$

Então, como pode ser observado na Figura 7, é aplicada uma função de ativação *g* a esse somatório para derivar a saída:

$$a_i = g(en_i) = g\left(\sum_{j=0}^{n} W_{ji} a_j\right)$$

É importante ressaltar que há a inclusão de parâmetro externo do neurônio artificial, um *bias* $W_{0,i}$, conectado a uma entrada fixa $a_0 = -1$. O termo $W_{0,i}$ define o limite real para a unidade, no

sentido de que a unidade é ativada quando a soma ponderada de entradas "reais" $\sum_{j=0}^{n} W_{ji} a_j$ excede $W_{0,i}$ (RUSSELL;NORVIG,2004).

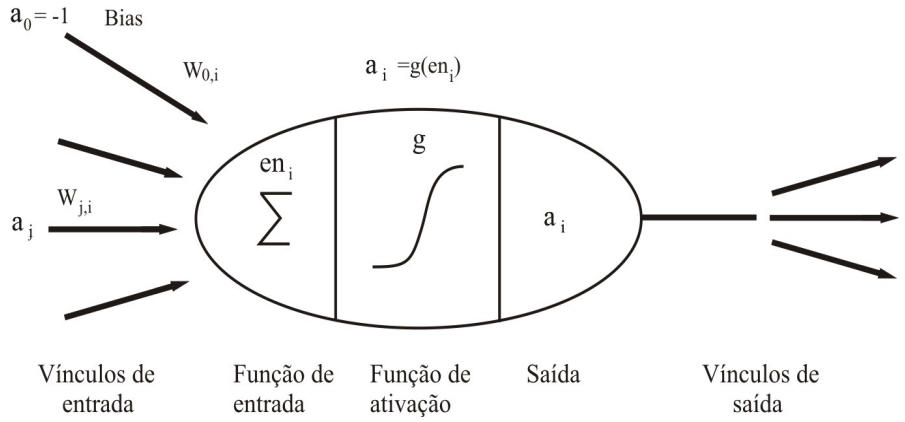

Figura 7: Modelo de um neurônio artificial. A ativação da saída da unidade é a_i e onde a_j é a ativação de saída da unidade *j* e W_{ji} é o peso no vínculo da unidade *j* até essa unidade (RUSSELL, 2003).

A arquitetura da RN utilizada nessa implementação é a *Perceptron Multilayer*. Essa caracteriza-se por possuir neurônios de entrada, neurônios ocultos e neurônios de saída. A função dos neurônios ocultos é intervir entre a entrada e a saída de maneira eficiente. As vantagens de adicionar camadas ocultas é que essas aumentam o número de hipóteses que a RN pode representar e, assim, é possível extrair estatísticas de ordem elevada. Entretanto, quando se tem uma RN com muitas camadas ocultas, estas são menos eficientes computacionalmente, já que necessitam de um maior tempo de processamento (RUSSELL;NORVIG,2004).

Para verificar a exatidão de uma RN, utiliza-se aqui, a técnica de validação cruzada ou *K-fold-cross-validation* (k-FCV) que consiste em dividir aleatoriamente o arquivo de padrões em *k* partes de mesmo tamanho. Assim, ocorre a geração dos arquivos para treinamento e validação. Inicialmente a sequência promotora, representada por nucleotídios, é dividida em 10 partes iguais, após é traduzida em números, sendo que para cada letra é atribuída uma sequência binária. No final da sequência é colocado o valor 1 indicando que é uma sequência promotora. Esse processo também é realizado para uma sequência gerada aleatoriamente, sendo que nesse caso o valor 0 é atribuido indicando que não é promotora. Esses valores são utilizados para o treinamento e testes da RN.

Após, a parte *k-ésima* da divisão da sequência promotora é unida com a parte *k-ésima* da divisão da sequência aleatória gerando a entrada de teste *k-ésima*, e assim sucessivamente para todos os *k* arquivos de testes. Já para as entradas do treino são concatenadas todas as partes menos a *k-ésima* parte da sequência promotora e todas as partes menos a *k-ésima* parte da sequência aleatória gerando assim o arquivo de treino *k-ésimo.* Esse processo é realizado para cada arquivo de treino. As etapas de treinamento e validação são repetidas *k* vezes, sendo utilizados para treinamento *k*-1 arquivos e para validação o *k-ésimo* arquivo não utilizado no treinamento. A cada interação, o arquivo de validação possui um *k* diferente (RUSSELL;NORVIG,2004).

A implementação sequencial do método de determinação de promotores através de uma RN que foi desenvolvido pela doutoranda do programa de Pós-Graduação em Biotecnologia da UCS Scheila de Avila e Silva, utilizando a linguagem R (SEEFELD, 2005), e pode ser visualizado na Seção A.5 do Anexo A.

2.4 Considerações Finais

Neste capítulo foram introduzidos os conceitos de biologia molecular necessários para o entendimento das pesquisas realizadas pelo grupo de bioinformática da UCS. Procurou-se resumir as informações necessárias e não discorrer a fundo o assunto sobre biologia molecular, já que este trabalho não se enquadra como uma pesquisa de bioinformática, e sim, como um trabalho de computação utilizando as ferramentas de bioinformática.

Nesse capítulo descreveu-se ainda os métodos de pesquisas utilizadas pelo grupo de bioinformática da UCS. Uma vez que são muitas as técnicas empregadas e em um volume grande de dados. O custo computacional requerido por essas pesquisas é elevado. Desta forma, a paralelização desses métodos pode contribuir de forma a diminuir o tempo de processamento.

A utilização de uma tecnologia de alto desempenho tem como principal objetivo dividir estes problemas de forma que esses possam ser executados em diferentes computadores simultaneamente. No próximo capítulo serão apresentados os principais conceitos sobre a computação distribuída e a computação paralela. Esses conceitos são apresentados de forma resumida e através da apresentação desses conceitos pretende-se familiarizar-se o leitor com arquiteturas paralelas, bem como, facilitar a compreensão das implementações desenvolvidas nesse trabalho.

3 COMPUTAÇÃO PARALELA

As ferramentas de bioinformática apresentadas no Capítulo 2 necessitam de uma alta capacidade de processamento. Isso deve-se principalmente, a grande quantidade de dados a serem processados e a realização de cálculos complexos. Desta forma a execução dessas ferramentas em um único computador é inviável. Por isso, existe a necessidade de utilizar-se uma tecnologia alternativa que permita diminuir o tempo computacional desse processamento. Dentre as alternativas existentes o uso de computação paralela é a mais indicada.

A computação paralela tem como principal objetivo a divisão de uma tarefa em partes menores e a distribuição destas partes entre diferentes processadores que trabalham de forma simultânea, ou seja, processam essas tarefas em paralelo. Contudo, essas unidades devem ser sincronizadas e se comunicar (DANTAS, 2005).

De acordo com a organização de memórias os computadores paralelos podem ser divididos em: arquiteturas com memória compartilhada ou arquiteturas com memória distribuída. As duas diferem, principalmente, porque em uma arquitetura de memória compartilhada as unidades de processamento utilizam um espaço de memórias com um endereçamento único. Já em uma arquitetura de memória distribuída cada unidade possui a sua própria memória e essas unidades se comunicam através de uma rede de comunicação. A Figura 8 ilustra essas diferentes arquiteturas. Essas duas arquiteturas são do tipo MIMD (*Multiple Instruction Multiple Data*) que consiste em múltiplos fluxos de instruções processando múltiplos conjuntos de dados (KONIGES, 2000).

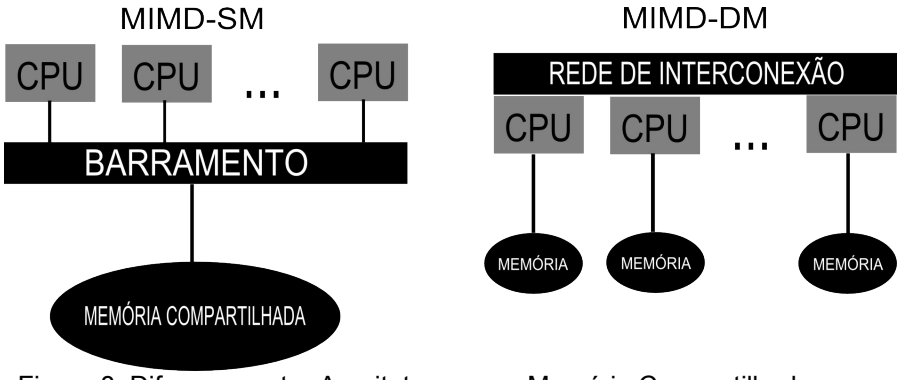

Figura 8: Diferença entre Arquiteturas com Memória Compartilhada e com Memória Distribuída.

Essas arquiteturas permitem a execução de múltiplas tarefas em paralelo, porém essas tarefas podem necessitar de uma comunicação e uma cooperação entre si. Nas próximas seções são descritos os conceitos relacionados às arquiteturas citadas anteriormente, bem como a forma de programação utilizadas nas mesmas.

3.1 Arquiteturas com Memória compartilhada

Quando uma arquitetura paralela é caracterizada por possuir uma memória compartilhada, refere-se ao fato dessa possuir compartilhamento global de uma única memória (ou conjunto de memórias) entre todas as unidades de processamento (DANTAS, 2005). Assim, existe um espaço de memória comum, utilizado para a troca de informações entre os processadores. O exemplo mais conhecido da arquitetura de memória compartilhada são os multiprocessadores. Nessa arquitetura todas as unidades de processamento utilizam um barramento comum para acessar a memória, como pode ser visualizado na Figura 8. Esse tipo de máquina possui apenas um espaço de endereçamento, de modo que todos os processadores podem endereçar todo o conjunto de memórias.

Uma das vantagens de se utilizar memória compartilhada é o acesso direto aos dados em memória utilizando-se apenas o barramento para o tráfego de informações. Em um barramento a latência é baixa e a largura de banda é usada ao máximo. Um desvantagem se refere a disputa pelo acesso a memória, pois todos os processadores podem necessitar do acesso a memória simultaneamente, o que torna o barramento um gargalo. Outra desvantagem refere-se a expansão do sistema, uma vez que com o aumento no número de processadores o sistema pode tornar-se mais lento devido ao aumento de disputa pelo barramento.

De acordo com os modos de acesso nos sistemas de memória compartilhada podem ser classificados como UMA (*uniform memory access* ou acesso uniforme a memória), onde o acesso a memória possui a mesma latência em relação a todos os processadores. Outro modo de acesso a memória consiste no NUMA (*non-uniform memory access* ou acesso não uniforme da memória), onde o tempo de acesso a memória para cada processador é diferente (KSHEMKALYANI;SINGHAL, 2008).

O modelo de programação mais utilizado em sistemas com memória compartilhada é *multithreanding* e as ferramentas mais comuns para esse modelo são as bibliotecas Pthreads e o OpenMP. Uma *thread* é um fluxo (concorrente) dentro de um mesmo processo. A utilização de múltiplas *threads* é uma maneira eficiente de explorar o paralelismo da arquitetura, pois diferentes *threads* podem rodar em diferentes processadores simultaneamente, compartilhando entre elas os recursos do processo, como por exemplo, o espaço de endereçamento (códigos e dados). Existem algumas vantagens na utilização de *threads* em relação a utilização de multiplos processos, como por exemplo a criação e o gerenciamento das *threads* é mais rápido que de processos. Porém a maior vantagem do uso de *threads* é a possibilidade da comunicação entre elas ser realizada utilizando a área de memória comum, através de operações de leitura e escrita (TANENBAUM;WOODHULL, 2000).

3.2 Arquiteturas com memória Distribuída

Em arquiteturas com memória distribuída, ou também conhecida como multicomputadores, as unidades de processamento são independentes sendo que cada uma delas possui sua própria memória. Essas unidades se comunicam através de um mecanismo de troca de mensagens por uma rede de computadores. Por isso são chamados também de sistemas de troca de mensagens (*message passing systems*) (DANTAS, 2005).

Uma vantagem dessa arquitetura é a escalabilidade fácil do sistema, pois pode-se aumentar a capacidade do processamento apenas adicionando computadores. Entre as principais desvantagens estão a necessidade de acessar informação de outro computador via rede, onde a latência é alta e a largura de banda é baixa se comparada a um ambiente com memória compartilhada. Além disso, existe a responsabilidade do programador em sincronizar o envio e recebimento de mensagens (DANTAS, 2005). Os exemplos mais conhecidos de arquiteturas com memória distribuída são os *clusters* e os *grids* computacionais. Os *grids* de computadores serão descritos com mais detalhes na próxima seção, uma vez que são a arquitetura escolhida para o desenvolvimento desse trabalho.

Um *cluster* de computadores pode ser definido como um conjunto de computadores dedicados ao processamento de uma tarefa. Geralmente os *clusters* de computadores são constituídos por computadores de uso pessoal (PCs), que são conectados por uma rede de comunicação de alta velocidade, formando uma plataforma para o processamento de aplicações paralelas (BUYYA, 1999). O tipo mais comum de *cluster* de computadores é o chamado *Beowulf*, projetado por Donald Becker da NASA (BEOWULF, 2009). Este tipo de *cluster* é constituído por nós de computadores pessoais utilizando o sistema operacional Linux e softwares para processamento paralelo como MPI (*Message Passing Interface*) e PVM (*Parallel Virtual Machine*) (BOOKMAN, 2003).

De acordo com a configuração dos computadores que compõem os *clusters* esses podem ser classificados em: os *clusters* homogêneos e os *clusters* heterogêneos. Os homogêneos são constituídos por computadores com a mesma arquitetura e o mesmo sistema operacional, assim não apresentam a necessidade de conversão de dados e nem atraso no processamento por dependência de um computador mais lento (DANTAS, 2005). Já heterogêneos são compostos por máquinas diferentes, tanto na arquitetura como no sistema operacional. O grande desafio é garantir que a heterogeneidade seja transparente para os usuários e que o uso de máquinas e pacotes de software compilados sobre diferentes sistemas operacionais executem de maneira correta (DANTAS, 2005).

A forma mais utilizada para a programação e sincronização de tarefas em *clusters* de computadores é o uso do paradigma de troca de mensagens. Existem diversas ferramentas que implementam esse paradigma, sendo que as mais conhecidas e utilizadas são: MPI e PVM.

3.3 Grids Computacionais

Como já mencionado anteriormente optou-se pela utilização de *grids* computacionais para o desenvolvimento desse trabalho. Optou-se pela utilização dessa arquitetura pois as tarefas a serem processadas pelos métodos descritos no Capítulo 2 são do tipo *bag-of-tasks*, ou seja, não necessitam de comunicação entre elas. Assim os grids computacionais tornam-se uma boa alternativa uma vez que constituem-se em uma plataforma de alto desempenho com baixo investimento. Para o desenvolvimento desse trabalho utilizou-se o GridUCS, que é um *grid* computacional formado por aproximadamente 300 computadores dos laboratórios de informática da UCS.

Os *grids* computacionais assemelham-se a *clusters* porém com algumas diferenças. Os *Grids* são um conjunto de computadores processando uma tarefa em comum dividida entre esses, porém esses computadores não são dedicados, ou seja, são computadores de uso comum e disponibilizam seu tempo ocioso de forma a concluir uma determinada tarefa. Além disso, a heterogeneidade dos componentes de um *grid* também é uma característica peculiar, assim como uma possível dispersão geográfica entre esses componentes. Existem diferentes ferramentas para o gerenciamento e a execução de tarefas em um *grid* computacional. Nas próximas seções serão abordadas as principais ferramentas utilizadas para o gerenciamento e utilização desse tipo de plataforma.

3.3.1 BOINC

O BOINC (*Berkeley Open Infrastructure for Network Computing*) é uma plataforma *open source* desenvolvida pelo *Space Sciences Laboratory of the University of California* com o objetivo de facilitar o desenvolvimento de aplicações para *grids* computacionais. Essa plataforma tem como objetivo obter recursos computacionais através da utilização da ociosidade de computadores fornecidos por voluntários. Esses são disponibilizados principalmente por usuários domésticos com PCs comuns e com acesso a internet.

Para a disponibilização do seu computador basta ao voluntário fazer o *download* de uma aplicação que atua como uma proteção de tela. Desta forma, quando o computador encontrar-se ocioso, a proteção de tela entra em ação e aloca os recursos do equipamento para o processamento das tarefas do *grid*. Nesse processamento as tarefas são realizadas com baixa prioridade, para que processos com maior prioridade não tenham seu desempenho prejudicado. Quando o usuário voltar a utilizar seu computador essa proteção de tela é interrompida e a execução da tarefa é concluída (ANDERSON, 2004).

Para incentivar o uso por parte dos voluntários o BOINC utiliza-se de um sistema de créditos, onde quanto mais recursos o usuários ceder para o *grid*, mais créditos ele irá acumular, sendo que os voluntários com mais créditos ganham destaque nos sites dos projetos. Nesses encontram-se publicados *rankings* com os voluntários que mais contribuem para os mesmos (ANDERSON, 2004).

Os projetos que são executados sobre o BOINC são independentes uns dos outros, sendo que cada um deles pode se encontrar em servidores distintos e sem nenhuma relação entre si. Entretanto, um usuário pode tornar-se voluntário em mais de um projeto e especificar a quantidade de recursos do seu equipamento que irá destinar a cada um dos projetos que participará (ANDERSON, 2004). Atualmente diversos projetos utilizam a plataforma BOINC entre esses destacam-se: *Einstein@Home*, *Folding@Home*, *LHC@Home*, *SETI@Home*, *Orbit@Home*.

3.3.2 *Globus*

O desenvolvimento do sistema *Globus* teve início no final de 1994 e é o resultado de pesquisas acadêmicas principalmente do *Argonne National Laboratory*, da *University of Southerm Califórnia* e da *University of Chicago*. Hoje diferentes instituições e empresas contribuem para o desenvolvimento do *Globus* formando a *Globus Alliance* (GLOBUS, 2009). Entre essas instituições e empresas pode-se destacar: IBM, *Microsoft*, NASA e DARPA (GLOBUS, 2009)

O *Globus* é um conjunto de ferramentas *open source* para o desenvolvimento de aplicações e sistemas de *grids* computacionais. Esse trata-se de um conjunto de softwares e bibliotecas destinados ao monitoramento, descoberta e gerência de recursos de um ambiente de *grid* (VARGAS; BARRETO, 2005), sendo que suas principais funcionalidades são:

1. Segurança: o *Globus* fornece autenticação e controle das permissões de usuários em relação aos recursos do *grid* através de certificados (sendo necessária apenas uma autenticação). Esse também possibilita o uso de criptografia e o controle de integridade no envio de mensagens (VARGAS; BARRETO, 2005).

2. Gerência de dados: esse possibilita serviços de transferência (baseado em FTP) de dados em paralelo, com o propósito de oferecer um melhor desempenho. Além disso, o Globus fornece acesso e integração de arquivos e bancos de dados, bem como replicação de dados em diversos nodos (GLOBUS, 2009).

3. Controle de execução de tarefas: com o *Globus* é possível efetuar o uso de recursos remotos para execução de tarefas no momento da solicitação (GLOBUS, 2009). Esse facilita a utilização desses recursos uma vez que possam existir dificuldades no gerenciamento de recursos de um *grid*, principalmente, pelo fato de que esses recursos poderem pertencer a organizações diferentes, as quais possuem suas próprias políticas de segurança;

4. Monitoramento de informação: o *Globus* permite obter informações e configurações dos recursos que compõem o *grid*. Como exemplo dessas informações cita-se: memória disponível, velocidade do processador, sistema operacional utilizado, dispositivos de armazenamento, base de dados, etc. Além disso o Globus permite localizar recursos com determinadas configurações. (GLOBUS, 2009);

5. Núcleo de execução: o *Globus* inclui ferramentas e bibliotecas que facilitam o desenvolvimento de novos serviços. Por exemplo utilizando-se essas bibliotecas pode-se criar um ambiente para a criação, execução e monitoramento de tarefas (GLOBUS, 2009).

6. Comunicacão: O *Globus* possui um serviço para melhorar a comunicação, o *Nexus*. Esse serviço é capaz de selecionar a melhor forma de comunicação entre os equipamentos do *grid (GLOBUS, 2009)*. Esse possui uma interface única, evitando que o programador tenha que se preocupar em trabalhar sob diferentes formas de comunicação dentro de sua aplicação, ou seja, o *Nexus* é capaz de selecionar a melhor forma de conexão. Por exemplo se os equipamentos estiverem geograficamente separados ele utilizara o protocolo TCP/IP, já se os equipamentos utilizarem memória compartilhada, ele fará a comunicação através da memória compartilhada (CIRNE; NETO, 2005).

Apesar de todas as funcionalidades oferecidas pelo Globus esse possui a desvantagem de necessitar de autenticação no momento de utilização de grid computacional, ou seja, um usuário que deseja acessar os recursos do *grid* computacional deverá se autenticar, ou negociar permissões com o administrador do domínio de recursos (ANDRADE;CIRNE;BRASILEIRO, 2003).

3.3.3 Ourgrid

O *Ourgrid* é uma ferramenta de *grid* computacional *open source* desenvolvido pela Universidade Federal de Campina Grande (UFCG) em parceria com a *Hewlett-Packard* (HP). Esse possui como característica a utilização da tecnologia *peer-to-peer* (P2P), e seu principal objetivo é a utilização do tempo ocioso de máquinas não dedicadas para o processamento de tarefas *Bag-of-Tasks*, isto é, tarefas que são independentes e não necessitam de comunicação entre elas.

No *Ourgrid* todos recursos de processamento pertencem a rede *Ourgrid*, e podem ser acessados por qualquer usuário da mesma (ANDRADE;CIRNE;BRASILEIRO, 2003). Esse utiliza-se de políticas de *Favors* para a distribuição de recursos, onde quanto mais recursos um usuário ceder a rede mais recursos ele receberá em troca de quem já utilizou da rede. Se um usuário não proceder desta forma o *Ourgrid* automaticamente, e gradualmente, retirará prioridades deste usuário, ficando em débito com a comunidade (ANDRADE;CIRNE;BRASILEIRO, 2003).

O software responsável pelo gerenciamento do *grid* é o *Ourgrid Toolkit* e sua estrutura do está dividida na seguinte forma (Figura 9):

1. *Mygrid*: componente que realiza o gerenciamento do *grid* e permite a interação do usuário com o *grid*. O *Mygrid* é responsável, principalmente, por conter as configurações do *grid*, e a distribuição das tarefas a serem executadas. É através do *Mygrid* que são gerenciadas as tarefas e os recursos do *grid* (ANDRADE *et* al.,2005).

2. *Ourgrid Peer*: é o componente que determina quais e como os computadores podem ser utilizados pelo *grid* na execução de uma tarefa. Quando esse recebe uma requisição ele gera

um conjunto de *grid machines* (recursos) que satisfaça essa requisição (ANDRADE *et* al.,2005). Caso o *peer* não possua recursos suficientes esse se comunica com os demais *peers* em busca de recursos.

3. *Ourgrid Community*: responsável por gerar a estrutura de recursos *Ourgrid* para serem usados pelo *Mygrid*. Esse é composto por *Ourgrid Peers* e é através dele que o usuário pode solicitar recursos computacionais na *Ourgrid Community* (ANDRADE *et* al.,2005).

4. *Swan*: os recursos que serão utilizados pelo *grid* necessitam de garantias que os usuários da *Ourgrid Community* não os danifiquem. Com esse objetivo foi criado o *Swan* que consiste em um serviço que é executado em cada máquina do *grid* e impede que tarefas mal intencionadas danifiquem o recurso ou acessem a rede de forma indevida. Todo processo é realizado utilizando máquinas virtuais isoladas do sistema operacional (ANDRADE *et* al.,2005).

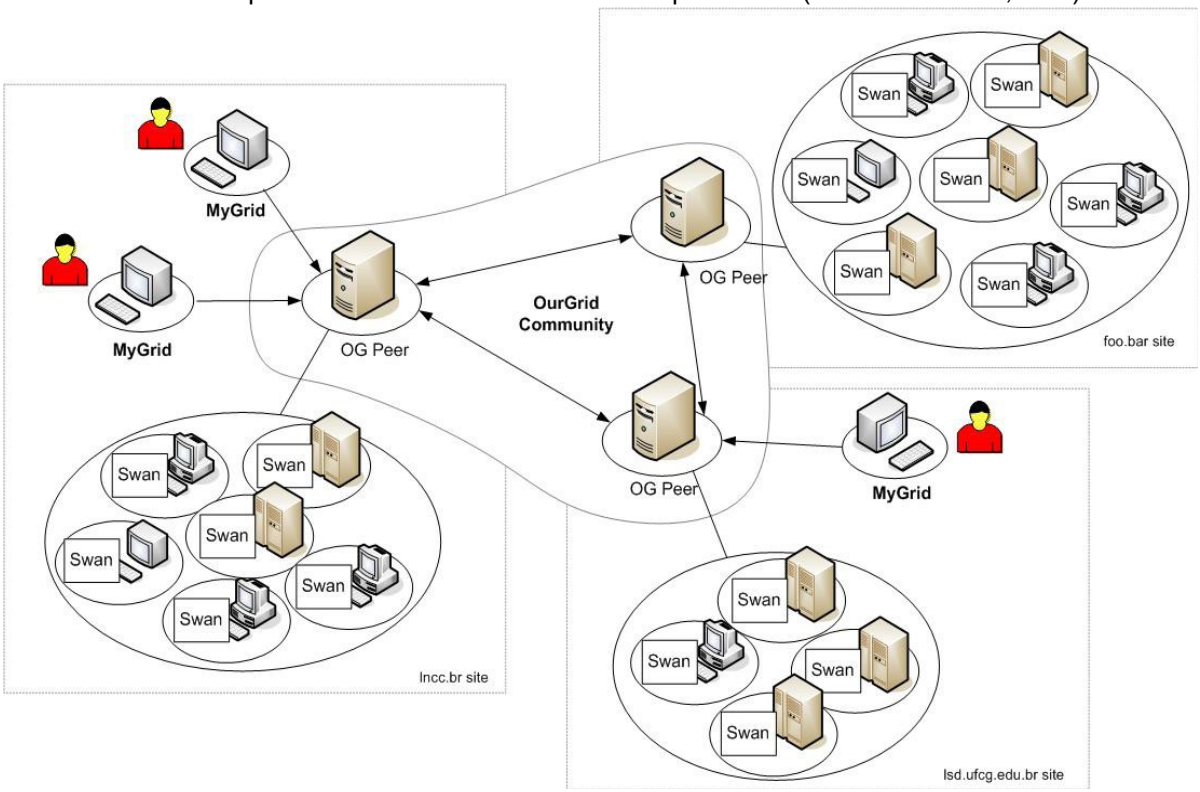

Figura 9: Arquitetura do sistema Ourgrid (ANDRADE *et* al.,2005)

De forma resumida, o funcionamento do Ourgrid é baseado em um protocolo de compartilhamento. Esse protocolo funciona do seguinte modo: cada *peer* possui dois serviços, que são o *Consumer* e o *Provider.* Quando um *Consumer* recebe uma requisição de um *Mygrid* por recursos, ele envia uma mensagem a toda rede P2P com as características desejadas. Esta mensagem então é recebida pelos *Providers* que são responsáveis por liberar os recursos. O *Provider* então responde, com outra mensagem, para o *Consumer*, informando se possui recursos disponíveis ou não. De posse dos recursos disponíveis o *Consumer* requisita através de uma mensagem que o *Mygrid* agende as tarefas em sua máquina. Finalmente, o *Mygrid* retorna ao *Consumer* uma mensagem contendo a lista de tarefas a ser executada (ANDRADE;CIRNE;BRASILEIRO, 2003).

O Ourgrid foi a ferramenta utilizada nesse trabalho, pois é a ferramenta que é utilizada no *grid* computacional da UCS (GridUCS). Optou-se também pela utilização do *Ourgrid* devido, principalmente pela simplicidade de configuração, gerenciamento e utilização do mesmo, se comparado com as demais ferramentas de gerenciamento de *grids* computacionais.

3.4 Considerações Finais

Neste capítulo foram introduzidos os conceitos sobre computação paralela. Inicialmente abordou-se conceitos sobre arquiteturas com memória compartilhada e arquitetura com memória distribuída. Como exemplo de arquiteturas com memória distribuída foram destacados os *clusters* e *grids*. Foram descritas ainda algumas ferramentas utilizadas para o gerenciamento de *grids* computacionais, que é a arquitetura utilizada para o desenvolvimento desse trabalho.

No próximo capítulo serão descritas as implementações desenvolvidas para a execução dos métodos descritos no Capítulo 2 em um *grid* computacional. Além disso, serão apresentados os resultados obtidos a partir da execução dessas implementações.

4 IMPLEMENTAÇÕES DESENVOLVIDAS E RESULTADOS OBTIDOS

Esse capítulo tem como objetivo descrever as implementações desenvolvidas no trabalho e apresentar os resultados obtidos. Essas implementações foram adaptações dos algoritmos sequenciais dos métodos descritos no Capítulo 2, de forma a executarem em *grid*. Na Figura 10 tem-se um fluxograma mostrando a estrutura básica de execução desses métodos. Os métodos de cálculo de entropia, cálculo de periodicidade e coeficiente de clusterização possuem como entrada uma sequência de DNA no formato Fasta (WHEELER et al, 2006) que podem ser obtidos a partir de repositórios públicos. Na Figura 11 tem-se um trecho de um arquivo do tipo Fasta.

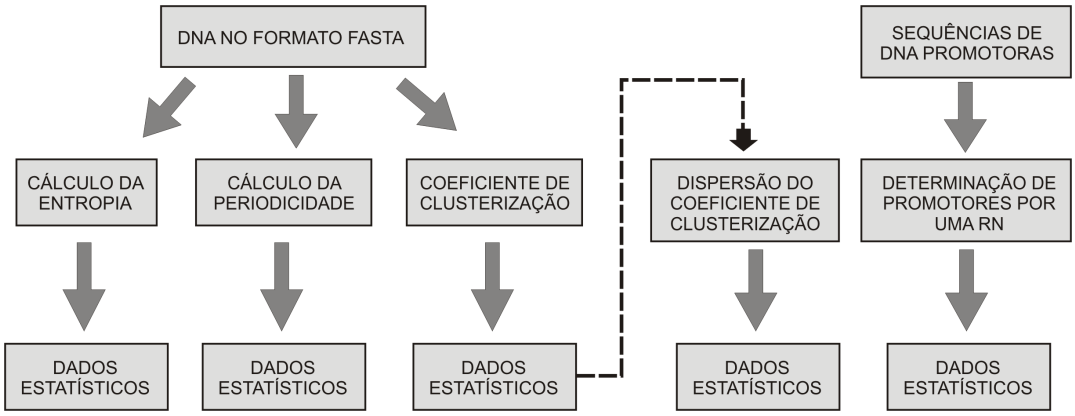

Figura 10: Fluxograma de execução dos métodos

Como pode ser visto na Figura 10, a partir dos arquivos de Fasta são executados os métodos de cálculo de entropia, cálculo de periodicidade, cálculo do coeficiente de clusterização. Os resultados gerados podem ser analisados por especialistas, ou ainda podem, serem utilizados como entrada para outros métodos. Por exemplo, a saída do método de cálculo coeficiente de clusterização é utilizado como entrada para o método de cálculo de dispersão do coeficiente de clusterização. Já o método de determinação de promotores através de uma RN, como pode ser visto na Figura 10, tem como entrada sequências promotoras, que podem ser obtidas através do repositório RegulonDB (RegulonDB, 2009), e possui como saída dados estatísticos para uma futura análise de especialistas.

```
1.>gi|257796365|gb|CP001734.1| Desulfohalobium retbaense DSM 5692, complete genome
2.ACAAGAGGAAAGTAGACGAGGATTGAGCTCATGCATCTGAAAAAGGTCAAGTCATCCATGGAGCCCCTTT
3.TTGAGCGGATGAAAACGGGTGTGGCCGACGGAGAACTCCCGAGTCATGACGATGTTTCTCAATTTGTTCG
4.TTTGTGTCGACAGATGTACGATTTTGCCGAGGAGGAGTGGTTGATGGAGGCTGAGGATTTCCTCCATCTG
5.GCCAACCAACTGTCTCGGGCGGTCAAAAATGGGGAACTTCCCGAGGTTGTCCAACTGGTGGACTCCCTGG
6.AAGATGCTTGCGCCTATTGCCACCGCACTTTCCGGGATTAGATCCTGTTTCTGAACCGCTTCGCTTGAGG
```

Figura 11: Um trecho de um arquivo no formato Fasta

4.1 Cálculo da entropia, periodicidade e coeficiente de clusterização

Uma vez que os métodos de cálculo de entropia, cálculo de periodicidade e cálculo do coeficiente de clusterização apresentam um mesmo fluxo de execução (Figura 11), as suas respectivas implementações para a execução em um *grid* computacional também foram semelhantes, e por isso serão detalhadas em uma mesma seção. Esses métodos possuem como entrada arquivos no formato Fasta (WHEELER et al, 2006), que geralmente possuem um grande número de pares de bases. Além disso, frequentemente, esses métodos são aplicados sobre vários arquivos de entrada.

De forma simplificada funcionamento do *grid* consiste de atribuir a cada máquina uma tarefa. Nesse caso isso pode ser realizado através da divisão dos arquivos Fasta em arquivos menores, ou seja, arquivos que contenham apenas uma parte do arquivo principal. Após a divisão essas partes são processadas em cada máquina do *grid*. Para a divisão dos arquivos de entrada foi desenvolvido um *script* na linguagem *Python.* Na Figura 12 tem-se um trecho desse *script*. Inicialmente esse *script* lista todos os arquivos Fasta do diretório corrente e em seguida os nomes desses arquivos são armazenados em uma lista (linha 1 e linha 2). Após essa lista é percorrida e os arquivos são divididos em partes menores, onde cada parte é identificada com o nome do original e com um número sequencial ao final. Essa divisão é baseada no número de linhas que cada arquivo parcial deve conter, sendo que esse valor é um parâmetro informado ao *script*.

```
1. lista_diretorio=commands.getoutput("ls")
2. lista_separada=lista_diretorio.split('\n')
3. n_lin = sys.argv[1]
4. for arq_fasta in lista_separada:
5.     if (arq_fasta[-5:]=='fasta'):
```

Figura 12: Trecho inicial do *script*.

Após essa divisão é gerado o arquivo a ser executado no *grid*. Para a geração desse arquivo foi desenvolvido um outro *script* em linguagem *Python*. Na Figura 13 tem-se um trecho desse *script*. Esse lista todos os arquivos resultantes da divisão realizada anteriormente (linha 2), e cada um desses arquivos é adicionado no arquivo de submissão do *grid*.

```
1. pedacos_fasta=open(arq_s_ext+'.txt','w+')
2. for i in range(0,gpos+1):
3.     script_grid.write('task:\n')
4.     script_grid.write('\t'+'init:\t'+'put '+ arq_s_ext +'_'+ str(i) +'.fasta ' +
arq_s_ext +'_'+ str(i) +'.fasta\n')
5.     script_grid.write('\t\t'+'put S_dna S_dna\n')
6.     script_grid.write('\t\t'+'put script_grid.sh script_grid.sh\n')
7.     script_grid.write('\t'+'remote: ./script_grid.sh ' + ' ' + arq_s_ext +
'_'+str(i) +'\n')
8.     script_grid.write('\t'+'final: get '+arq_s_ext+'_'+str(i)+'.tar.gz '+
arq_s_ext+'_'+str(i)+'.tar.gz\n')
9.     pedacos_fasta.write(arq_s_ext+'_'+str(i)+'\n')
10.pedacos_fasta.close()
```

Figura 13: Trecho gerador do arquivo a ser submetido no *grid*.

Na Figura 14 tem-se um exemplo de um arquivo de submissão de tarefas. O início do arquivo é indicado com a palavra reservada *job*, e o nome da execução. Esse nome é atribuído através da palavra reservada *label* (linha 2). O trecho de código de uma tarefa a ser executada no *grid* é iniciada através das palavras reservadas *task* e *init* (linha 3 e linha 4). O comando *put* corresponde aos arquivos a serem enviados aos computadores do *grid* (linhas 4, 5 e 6), sendo que o primeiro atributo do comando *put* corresponde ao nome do arquivo a ser enviado e o segundo ao nome que esse arquivo receberá na máquina remota (computador do *grid*). Já o comando *remote* corresponde ao comando a ser executado nos computadores do *grid* (linha 7). Finalmente, com os comandos *final* e *get*, os arquivos são transferidos da máquina do *grid* para a máquina de submissão (linha 8). No comando get o primeiro atributo corresponde ao nome do arquivo a ser buscado na máquina do *grid*, e o segundo atributo corresponde ao nome que este arquivo receberá ao retornar.

```
1. job:
2.     label: Mauricio
3. task:
4.     init:put Kineococcus_radiotolerans_SRS30216_0.fasta
Kineococcus_radiotolerans_SRS30216_0.fasta
5.           put S_dna S_dna
6.           put script_grid.sh script_grid.sh
7.     remote: ./script_grid.sh 4096 Kineococcus_radiotolerans_SRS30216_0
8.     final: get Kineococcus_radiotolerans_SRS30216_0.s
Kineococcus_radiotolerans_SRS30216_0.s
```

Figura 14: Trecho do arquivo de submissão no *grid*.

Após a execução das tarefas no *grid* um outro *script*, também implementado na linguagem *Python*, é responsável por unir todos os arquivos de resultados. Um trecho do código responsável por essa tarefa é dado na Figura 15. Inicialmente é criado um diretório onde serão armazenados os arquivos com os resultados (linha 1). Após são listados todos os arquivos de resultado que possuem o nome do arquivo original Fasta (linha 2). Esses arquivos são abertos e o conteúdo desses é armazenado em uma lista, onde cada posição dessa lista corresponde a uma linha dos arquivos de resultado (linha 3 a linha 6). Após esse arquivo de resultado é removido (linha 7). Por fim, a lista com os resultados é gravada em um arquivo de saída, e desta forma, concatenando todos os arquivos de resultado (linhas 8 e 9). Ao final esse arquivo é movido para o diretório criado para armazenar os resultados (linha 10).

```
1. res=commands.getoutput("mkdir resultado")
2. for i in range(0,n_arq):
3.     arquivo=open(arq_orig+'_'+str(i)+'_'+str(size)+'.s','r')
4.     linhas=arquivo.readlines()
5.     linhas=[line.strip() for line in linhas]
6.     arquivo.close()
7.     res=commands.getoutput("rm "+arq_orig+'_'+str(i)+'_'+str(size)+'.s')
8.     for l in linhas:
9.           saida.write(l+'\n')
10.res=commands.getoutput("mv "+arquivo_orig+'_'+str(size)+'.clu resultado')
```

Figura 15: *Script* responsável por unir as partes do resultado.

4.2 Cálculo da dispersão do coeficiente de clusterização

O fluxo de execução desse método é semelhante aos anteriores, mas com algumas particularidades. Uma diferença consiste no fato que esse recebe como entrada os dados gerados pelo método de cálculo de coeficiente de clusterização, e não um arquivo no formato Fasta. Porém, o principal motivo para a realização de modificações, no código fonte do método, é que esse possui como saída a média dos valores de dispersão de cada um dos grafos. Nesse caso, foi necessário alterá-lo de forma que cada tarefa executada no *grid* não gerasse como saída a média mas sim, apenas a soma destes valores. A média será calculada posteriormente.

Inicialmente o *script* gerador do arquivo a ser submetido no *grid* realiza os mesmos procedimentos que os métodos anteriores porém com algumas diferenças. Primeiramente não é realizada uma divisão dos arquivos de entrada, uma vez que esses são os resultados não unidos do processamento do método de cálculo de coeficiente de clusterização. Nesse *script*, como pode ser visto na Figura 16 os arquivos de entrada são listados (linha 1) e armazenados (linha 2) em uma lista. Após essa lista é percorrida e os arquivos de entrada (extensão .clu) são filtrados e adicionados no arquivo de submissão do *grid* (linha 3).

```
1. lista_diretorio=commands.getoutput("ls")
2. lista_separada=lista_diretorio.split('\n')
...
3. script_grid.write('\t'+'init:\t'+'put '+ arq_s_ext + '.clu ' + arq_s_ext
+'.clu\n')
...
```

Figura 16: Trecho inicial do *script*.

Após o processamento das tarefas no *grid* é executado um outro *script* (Figura 17) que realiza o processo de unir os resultados. Nesse os valores resultantes de cada arquivo são somados, e ao final é calculada a média (coeficiente de dispersão). Após, a média calculada é gravada em um arquivo de saída (linha 18).

```
1. total_col0=0.0
2. total_col1=0.0
3. total_linhas=0
4. for i in range(0,n_arq):
5.     arq_nlin=open(arquivo_orig+'_'+str(i)+'_'+str(size)+'.clu','r')
6.     nlin=arq_nlin.readlines()
7.     total_linhas=total_linhas+len(nlin)
8.     arq_nlin.close()
9.     arquivo=open(arquivo_orig+'_'+str(i)+'_'+str(size)+'_'+str(jan)+'.s','r')
10.    linhas=arquivo.readlines()
11.    for l in linhas:
12.          ln=l.split(' ')
13.          total_col0=total_col0+float(ln[0])
14.          total_col1=total_col1+float(ln[1])
15.    arquivo.close()
16.    res=commands.getoutput("rm "+arquivo_orig+'_'+str(i)+'_'+str(size)
+'_'+str(jan)+'.s')
17. saida=open(arquivo_orig+'_'+str(size)+'_'+str(jan)+'.s','w+')
18. saida.write(str(total_col0/total_linhas)+' '+str(total_col1/total_linhas))
```

Figura 17: Trecho gerador da média.

4.3 Determinação de promotores através de uma rede neural

Essa implementação baseia-se em um algoritmo sequencial já disponível, que foi desenvolvido pela doutoranda do programa de Pós-Graduação em Biotecnologia da UCS Scheila de Avila e Silva, utilizando a linguagem R (SEEFELD, 2005). Esse algoritmo, na versão sequencial, possui quatro laços encadeados, como pode ser visualizado na Figura 18. O laço mais externo (linha 1) refere-se ao número de neurônios na camada oculta. O laço subsequente (linha 2) é referente ao número da rede. O terceiro laço (linha 3) refere-se ao número de épocas utilizadas na RN. O laço mais interno (linha 4) refere-se ao número de listas inicializadas (arquivos de entrada). Já o bloco de comandos corresponde aos comandos que realizam as simulações da RN.

```
...
1. for Noculto=1 to 8 do
2.     for i=1 to 10 do
3.            for j=1 to 30 do
4.                   for k=1 to 10
5.                          Bloco de comandos
...
```

Figura 18: Laços encadeados.

Uma vez que cada laço é independente pode-se executar cada iteração (bloco de comandos) desses laços separadamente. Para isso basta realizar as iterações no *script* responsável por gerar os arquivos de submissão de tarefas ao *grid*. Desta forma, tem-se a execução de cada iteração em computador diferente do *grid*.

Nessa implementação foram realizados 3 testes. No primeiro deles os dois laços mais externos foram divididos de forma a serem distribuídos no *grid*. Nesse caso os dois laços mais internos são executados em cada máquina do *grid*. No segundo teste apenas o laço mais interno foi mantido para ser executado em cada máquina do *grid*. Assim dividiu-se os três laços mais externos, para serem executados de forma distribuída. O último teste dividiu-se os quatro laços para serem executados de forma distribuída, mantendo-se assim apenas o bloco de comandos responsável por realizar a RN em cada máquina do *grid*. Para a distribuição das tarefas foi implementado um *script* escrito na linguagem *Python*, sendo que na Figura 19 tem-se um trecho desse *script*.

```
...
1. for oculto in range(1,9):
2.     for i in range(1,11):
3.         for j in range(1,31):
4.             for k in range(1,11):
5.                 iteracoes=j*5
6.                 script_grid.write('task:\n')
7.                 script_grid.write('\t'+'init:\t'+'put
TreinoRNDGProm_Sigma70Aleat1 TreinoRNDGProm_Sigma70Aleat1\n')
8.                 script_grid.write('\t\t'+'put TesteRNDGProm_Sigma70Aleat1
TesteRNDGProm_Sigma70Aleat1\n')
...
9.                 script_grid.write('\t\t'+'put script_grid.sh script_grid.sh\n')
10.                 script_grid.write('\t\t'+'put Script.R Script.R\n')
11.                 script_grid.write('\t'+'remote: ./script_grid.sh '+ str(oculto)
+ ' ' + str(i) + ' '+ str(iteracoes) + ' ' + str(K) + '\n')
12.                 script_grid.write('\t'+'final: get arquivo_'+ str(oculto) +'_'+
str(i) +'_'+ str(iteracoes) + '_' + str(k) +'.tar.gz' + ' arquivo_'+ str(oculto)
+'_'+ str(i) +'_'+ str(iteracoes) + '_' + str(k) +'.tar.gz\n')
...
```

Figura 19: Trecho do *script* gerador do arquivo de submissão.

A execução do algoritmo em R em cada computador do *grid* é feita através de um *script* desenvolvido utilizando-se a linguagem *Shell Script*. Esse tem como parâmetro o número de laços a serem particionados e ao final esse compacta os arquivos obtidos a partir da execução da RN. Esses arquivos possuem a taxa de acerto da RN, ou seja, valores entre 0 e 1 que correspondem a probabilidade dessa sequência de DNA ser ou não promotora. Também é gerado o erro médio quadratico a fim de verificar a qualidade dos resultados obtidos. Na Figura 20 tem-se o código executado em cada máquina do *grid*.

```
1. /opt/R-2.9.0/bin/R --file=Script.R --args $1 $2 $3 $4
2. tar -czf arquivo_$1_$2_$3_$4.tar.gz *.txt
```

Figura 20: Código a ser executado no *grid*.

4.4 Resultados obtidos

Para os testes das implementações utilizou-se o Griducs que consiste em um *grid* institucional existente na UCS. Esse é formado, por aproximadamente, 300 computadores não dedicados, isto é, computadores que possuem como finalidade principal o uso em atividades didáticas em cursos de Graduação e Pós-Graduação da UCS. Esses computadores possuem 3 Sistemas Operacionais: Windows XP e Linux Ubuntu que são utilizados por alunos em atividades de ensino; e o Linux Ubuntu 6.12 Server, que é a distribuição onde estão instaladas as ferramentas necessárias para disponibilizar os recursos do equipamento para o *grid* computacional. Além dos 300 computadores não dedicados fazem parte ainda desse *grid* computacional 2 clusters de computadores. Porém esses não foram utilizados nos testes realizados nesse trabalho.

Para a execução dos métodos de cálculo de entropia, periodicidade, coeficiente de clusterização e dispersão do coeficiente de clusterização foram utilizados como entrada todos os arquivos Fasta correspondentes as espécies do *Kingdom Bacteria* e do *Kingdom Fungi* disponíveis no NCBI até a data de 26 de outubro de 2009 (WHEELER et al, 2006). Para o método de predição de

promotores através de uma RN foram utilizados todos os promotores disponíveis no RegulonDB (RegulonDB, 2009) associados ao Sigma70 (peso de sedimentação) até a mesma data.

Para a análise das implementações desenvolvidas calculou-se o *speed-up* e a eficiência das implementações desenvolvidas. O *speed-up* (S_p) tem como objetivo demonstrar quanto o a execução paralela é mais rápida que a execução sequencial e seu cálculo é feito através da divisão do tempo sequencial (T_s) pelo tempo paralelo (T_p), como segue T_s / T_p . O *speed-up* ideal é obtido quando seu resultado é igual ao número de máquinas ($S_p = p$). Já a eficiência refere-se percentual de uso dos processadores na execução, o valor dessa varia de 0 a 1, e é calculada pela divisão do *speed-up* (S_p) pelo número de processadores (p) utilizados, como segue S_p / p .

Na Figura 21 tem-se o tempo sequencial e o tempo paralelo dos métodos. O tempo sequencial é uma estimativa baseada na soma de todas as tarefas executadas no *grid.* Optou-se pelo cálculo dessa estimativa pois a execução sequencial de todos os métodos com um único computador era inviável. Nos testes dos cálculos de entropia, periodicidade, e coeficiente de clusterização foram utilizadas 177 máquinas. Já para o cálculo da dispersão do coeficiente de clusterização foram utilizadas 142 máquinas.O número de máquinas utilizadas nesses testes correspondem ao número de máquinas disponíveis no *grid* no momento dos testes. Para a predição de promotores através de uma RN foram utilizadas 80 máquinas uma vez que esse era o número máximo de tarefas existentes.

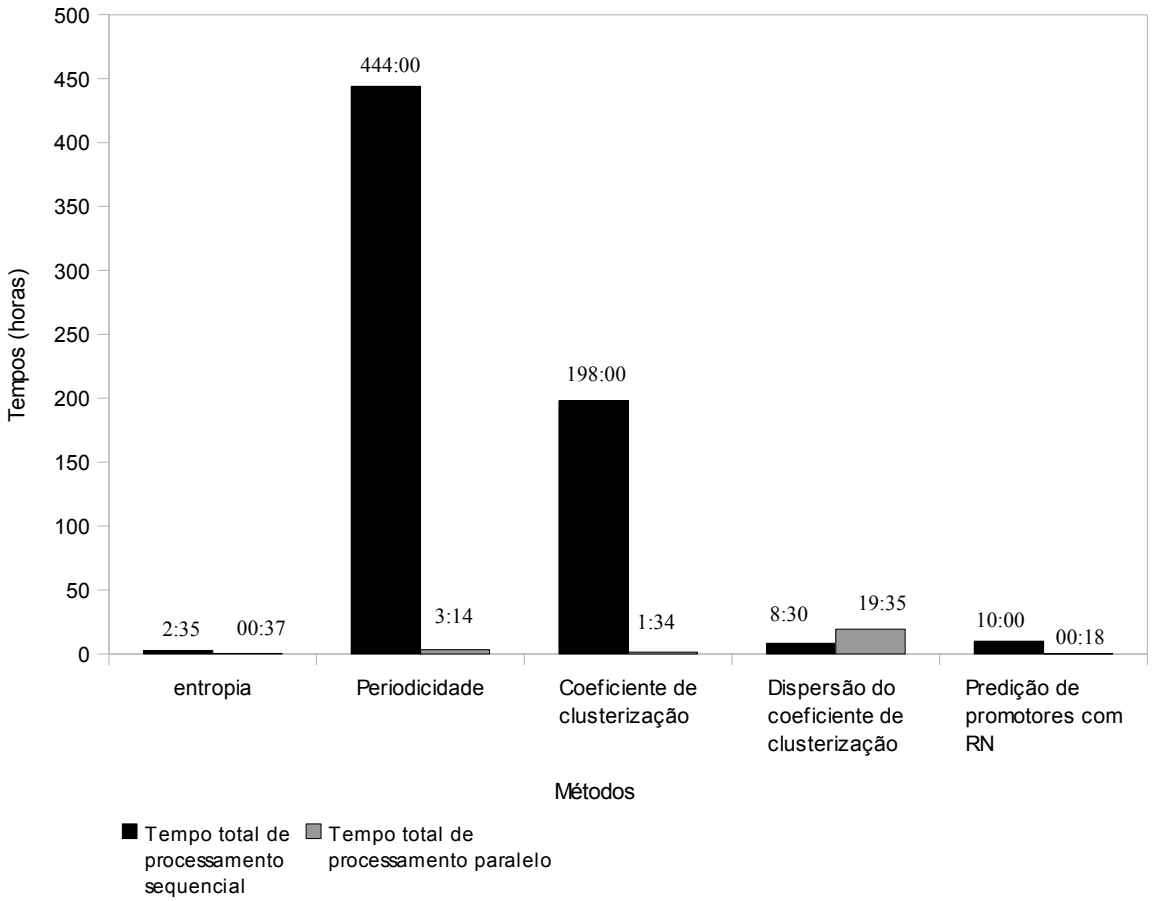

Figura 21: Tempos de processamentos sequencial e paralelo dos métodos

Para o cálculo de entropia pode-se observar que o tempo de sequencial é um pouco superior ao tempo paralelo. Como pode ser visto na Figura 22, o *speed-up* obtido nesse método é de apenas 4,49 e como pode ser observado na Figura 23 tem uma eficiência de apenas 0,025. Isso deve-se ao fato que o tempo gasto na fase de execução de cada tarefa, em média, é de 1,3 segundos, já o tempo de transferência de dados para cada tarefa é em média de 2 segundos, assim, tem-se um tempo elevado de transferência se comparado ao tempo de execução, que acarreta em um baixo desempenho da implementação desenvolvida.

No método de cálculo de periodicidade pode-se observar na Figura 21 que o tempo sequencial é muito superior que o tempo paralelo, existindo assim uma melhora significativa no tempo processamento. Na Figura 22 pode ser observado que o *speed-up* obtido nesse método é de 137,96, o que significa uma grande melhora durante a execução paralela em relação a sequencial. E pode ser observado na Figura 23 que esse tem uma eficiência de 0,779 o que significa que foi utilizado um percentual elevado dos recursos de cada computador do *grid* na execução. Isso deve-se ao fato que esse método utiliza 0,8 segundos de transferência de dados para cada tarefa, o que é baixo comparado aos 205 segundos utilizados para execução de cada tarefa.

Já para o cálculo do coeficiente de clusterização, como pode ser observado na Figura 21, tem-se uma melhora significativa no tempo processamento do método paralelo em relação ao sequencial. Na Figura 22 pode-se observar que o *speed-up* desse método é de 126,87. Já na Figura 23 pode ser observado que a eficiência também é elevada com valor de 0,716. Neste caso o tempo de processamento de cada tarefa é de 92 segundos e o tempo de transferência de dados 1,2 segundos.

Para o método da dispersão do coeficiente de clusterização como visto na Figura 21 o tempo de execução paralelo é maior que o tempo de execução sequencial. Como pode ser visto na Figura 22, o *speed-up* obtido nesse método é de apenas 0,43 e como pode ser observado na Figura 23 tem uma eficiência de apenas 0,003. Isso pode ser justificado pois o tempo de transferência de dados para cada máquina é de 1,5 segundos, muito alto se comparado ao tempo de execução de apenas 0,6 segundos para cada tarefa, Assim tem-se um tempo muito elevado de transferência de dados. Essa execução poderia ser melhorada através da execução de arquivos maiores, ou seja, na divisão dos arquivos Fasta originais em arquivos maiores. Assim, diminuindo o número de tarefas da execução e aumentando o tempo necessário para o processamento cada tarefa nos nodos do *grid*.

Para o método de predição de promotores através de uma RN, como pode ser observado na Figura 21 o tempo de execução paralelo é consideravelmente menor que o tempo de execução sequencial. Na Figura 22 e Figura 23 pode observar-se que o *speed-up* e *a* eficiência desse método são de 33,66 e de 0,420, respectivamente. Esse método obteve bons resultados, pois o tempo de transferência de dados, é de 29 segundos, sendo mais baixo que de execução, que é 45 segundos. Nesse método foram feitos três testes. Porém dois testes, o da paralelização de 3 laços e o da paralelização de 4 laços apresentaram resultados de tempo muito elevados, e neste caso optou-se por abortar a execução. Neste caso, tem-se um tempo de transferência de arquivos (envio e recebimento) superior ao tempo de execução.

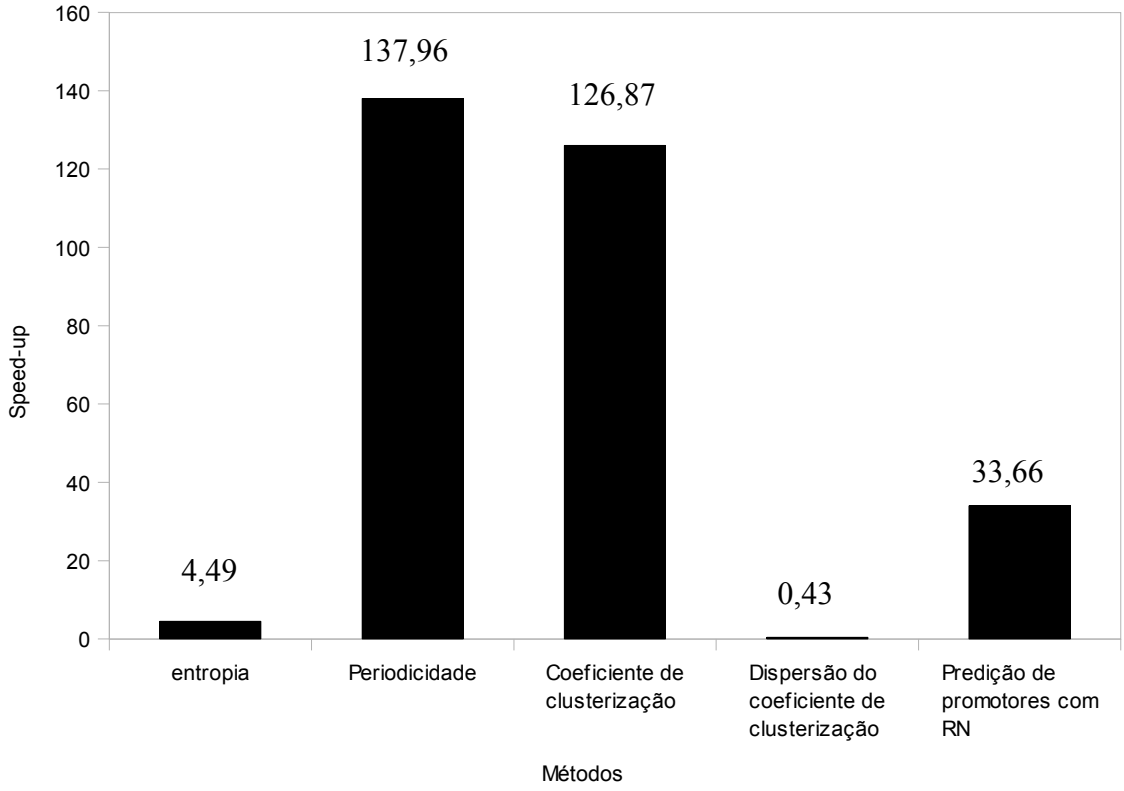

Figura 22: Speed-up dos métodos

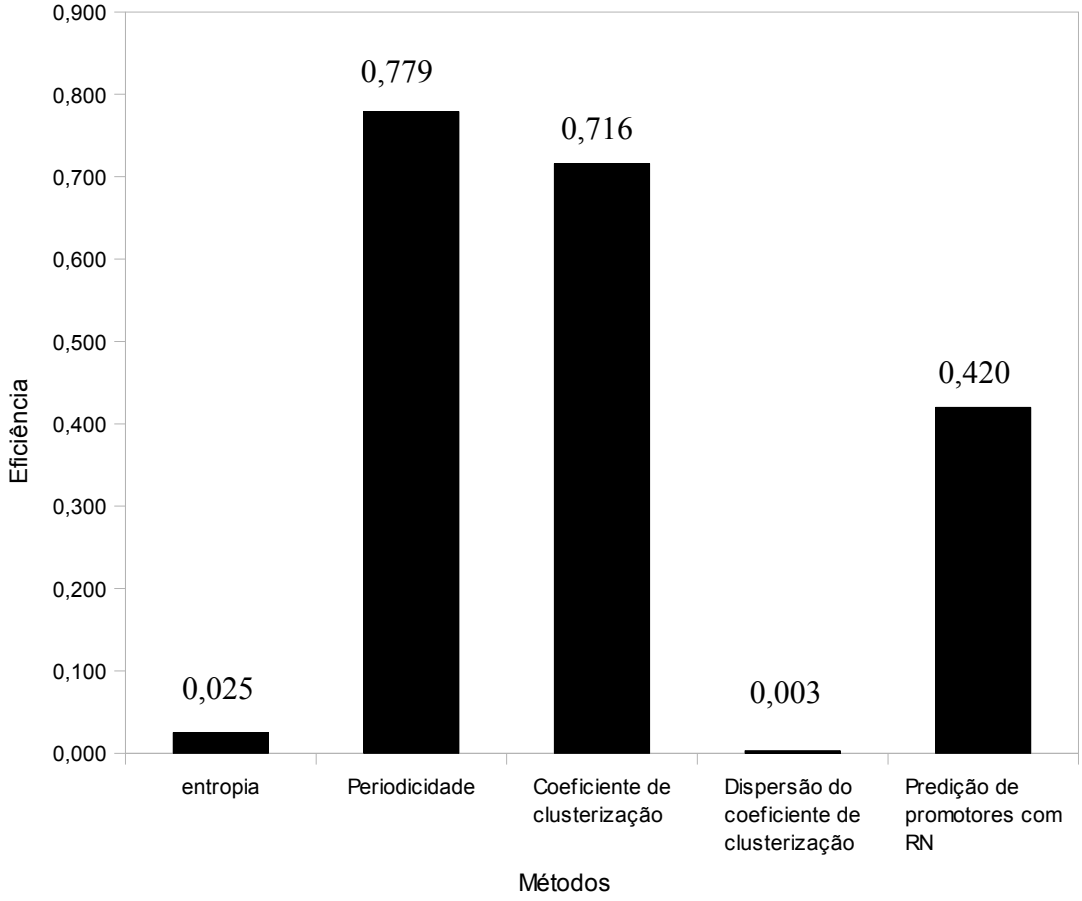

Figura 23: Eficiência dos métodos

O método de predição de promotores através de uma RN foi tema de uma pesquisa coordenada pelo Prof. Dr. Sérgio Echeverrigaray do Centro de Biologia da UCS. Essa pesquisa resultou na participação do aluno Mauricio Adami Mariani, autor desse trabalho, no evento XVII Encontro de Jovens Pesquisadores da UCS de 2009 como pesquisador voluntário. A pesquisa foi intitulada de: “Utilização de Grids Computacionais para a Paralelização de um Algoritmo de Aprendizado de Máquina” (MARIANI *et* al., 2009). Essa ainda teve a participação dos colaboradores Prof. Dr. Günther J.L. Gerhardt, doutoranda Scheila de Avila e Silva e do Prof. Msc. André Martinotto. O resumo apresentado no XVII Encontro de Jovens Pesquisadores da UCS encontra-se no Anexo B.

5 CONCLUSÃO

Nesse trabalho estudou-se conceitos básicos de biologia molecular, assim como os métodos de bioinformática do grupo de pesquisa da UCS gerenciado pelo Prof. Dr. Günther J. L. Gerhardt. A área da biologia apresenta uma grande campo de pesquisas e uma grande quantidade de dados para serem processados. Com o auxilio da computação essas tarefas podem ser realizadas de forma mais rápida e eficiente. Esse estudo foi realizado com o objetivo de utilizar esses métodos em um ambiente de alto desempenho. Optou-se pelos *grids* computacionais esses uma alternativa para a obtenção de um alto poder computacional a um baixo custo.

O ambiente de *grid* computacional escolhido para a execução dos métodos foi o *Ourgrid*. Esse possui como vantagens principalmente a simplicidade de configuração, de gerenciamento e de utilização. Além disso foi escolhido por já ser a plataforma utilizada no GridUCS. Através do *Ourgrid* foi possível paralelizar os cinco métodos propostos.

Para a paralelização dos métodos de cálculo de entropia, cálculo de periodicidade, cálculo do coeficiente de clusterização e cálculo da dispersão do coeficiente de clusterização optou-se pela divisão dos arquivos Fasta utilizados como entrada. Já para o método de predição de promotores através de uma RN a metodologia de execução consistiu na distribuição dos laços do algoritmo sequencial. Com esse estudo pode-se obter os tempos relativos ao processamento dos métodos em um ambiente de alto desempenho.

Das implementações desenvolvidas três dessas execuções paralelas (cálculo de periodicidade, cálculo do coeficiente de clusterização e predição de promotores com uma RN) demonstraram-se mais rápidas que a suas respectivas execuções sequenciais. Uma alternativa para diminuir ainda mais o tempo de processamento de dois desses métodos (cálculo de periodicidade e cálculo do coeficiente de clusterização) seria o aumento no número de clientes do *grid* computacional. O terceiro método que obteve melhora no seu tempo de processamento (predição de promotores com uma RN), obteve essa melhora através da distribuição de dois laços, gerando 80 tarefas, como comentado na Seção 4.4 tratou-se, ainda uma distribuição de 3 e 4 laços que compõem o programa sequencial, porém esses processamentos obtiveram tempos muito elevados, abortando-se as respectivas execuções.

Já os métodos cálculo de entropia e dispersão do cálculo do coeficiente de clusterização não apresentaram bons resultados, não havendo melhoras significativas nos seus respectivos tempos de processamento ou até mesmo um menor desempenho. Esses utilizaram mais tempo para a transferência dos dados para os computadores do *grid* do que no processamento das sequências. Uma alternativa para melhora de seus tempos de processamento seria de dividir os arquivos de entrada em partes maiores, gerando menos arquivos, e desta forma, obtendo-se uma melhor relação entre o tempo de transferência de dados e o tempo de processamento seria mais adequada.

Tendo em vista os objetivos iniciais deste trabalho, pode-se afirmar que eles foram concluídos. Além dos estudos e testes realizados, o resultado deste trabalho são *scripts* que podem ser usados para a utilização dos métodos em *Ourgrid* em outros conjuntos de dados.

5.1 Trabalhos Futuros

Este trabalho teve como objetivo auxiliar as pesquisas em bioinformática da UCS. A partir das implementações desenvolvidas procurou-se diminuir o tempo de processamento de alguns métodos utilizados nessa área. A bioinformática é uma área que possui diversos métodos que necessitam de processamento de alto desempenho. Assim pode-se relacionar algumas possibilidades de trabalhos futuros a seguir:

- Adaptação de outros métodos da bioinformática para plataformas de alto desempenho.
- Expansão da capacidade do GridUCS através da utilização de computadores que não estejam localizados na UCS, e sim dispersos geograficamente, ou ainda através de computadores disponibilizados por voluntários.
- Criação de um ambiente *web* que permita a disponibilização das ferramentas desenvolvidas para que possam ser utilizadas para um processamento em tempo real.
- A criação de um ambiente *web* para a disponibilização dos resultados gerados pelas execuções desse trabalho.

REFERÊNCIAS

ANDERSON, D. P. BOINC: A System for Public-Resource Computing and Storage. 5th IEEE/ACM International Workshop on Grid Computing, v.1, n.1, Sept./Nov. 2004.

ANDRADE, N.;CIRNE W.;BRASILEIRO, F.; OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing. Universidade Federal de Campina Grande, Campina Grande, 2003.

ANDRADE, N.;COSTA, L .;GERMÓGLIO, G.;CIRNE, W.; Peer-to-peer grid computing with the OurGrid Community. Universidade Federal de Campina Grande, Campina Grande, 2005.

BEOWULF. Beowulf.org The Beowulf Cluster site . Disponível em: <http://www.beowulf.org/>; Acesso em: setembro. 2009

BOOKMAN, C.; Agrupamento de Computadores em Linux. Rio de Janeiro, Ciência Moderna, 2003.

BUYYA, R. High Performance Cluster Computing: Architectures and Systems. [S.l.]: Prentice Hall, 1999.

CIRNE, W.; NETO, E. S. Grids Computacionais: Da Computação de Alto Desempenho a Serviços Sob Demanda. Fortaleza, CE: SBC, Universidade Federal do Ceara, Universidade Estadual do Ceara, 2005.

DANTAS, M. Computacão distribuída de alto desempenho. Rio de Janeiro: Axcel Books, 2005.

DONGARRA, J.;FOSTER, I.; FOX, G.;GROPP, W.;KENNEDY K.; TORCZON, L.;WHITE, A. ; Sourcebook of Parallel Computing. San Franscisco, California; Elsevier Science; 2003.

FARACH, M.; NOORDEWIER M.; SAVARI S.; SHEPP L.; WYNER A.; ZIV J.; On the entropy of DNA: Algorithms and measurements based on memory and rapid convergence. Proc. 5th Annual ACM-SIAM symposium on Discrete Algorithms; ACM-SIAM, 1994.

FOSTER Ian. Designing and Building Parallel Programs. 1.ed. Reading, USA: Addison Wesley, 1995.

GERHARDT ,G. J. L.; PANIS, R. J.; SANTOS, L. Presença de Períodos em Sequências de DNA. Caxias do Sul: UCS, Artigo não publicado, ano 2004.

GERHARDT ,G. J.L.; LEMKE, N.; CORSO,G.;Network clustering coefficient approach to DNA sequence analysis;Artigo publicado em Chaos, Solitons and Fractals,2005;

GERHARDT, G. J. L., SILVA, S. de A.; PANIS, R. J.; SANTOS, L. dos; Correlações entre Cromossomos Usando Periodicidades em Seqüências de DNA. II Workshop de Tecn. da Inf. aplicada ao Meio Ambiente; Publicado em 2004.

GIBAS, C.; JAMBECK, P.;Developing Bioinformatics Computer Skills; Estados Unidos, O'Reilly, 2001.

GLOBUS. The Globus Alliance. Disponvel em: <http://www.globus.org/>. Acesso em: julho. 2009.

HAYKIN, S.. Neural networks: a comprehensive foundation. 2. ed. New Jersey: Prentice-Hall, 2005.

HORTA, J. O. C.; Estimadores de Entropia para Sequências de DNA; 2001; 144p. Dissertação (Trabalho de Conclusão de Mestrado em Matemática Aplicada) – Universidade de São Paulo, São Paulo.

JACOB, B.; BROWN, M.; FUKUI, K.; TRIVEDI, N. Introduction to Grid Computing. , 2005. Disponvel em: http://www.redbooks.ibm.com/abstracts/sg246778.html. Acesso em: outubro de 2009.

KONIGES, A. E.; Industrial Strength Parallel Computing;San Francisco, California, Morgan Kaufmann Publishers, 2000.

KSHEMKALYANI, A. D. ; SINGHAL M.; Distributed Computing Principles, Algorithms, and Systems; New York; Cambridge University Press; 2008.

LESK, Arthur M., Introdução à Bioinformática, 2ª Ed.- Porto Alegre: Artmed, 2008.

LEWIN, B.;Genes VII;1.ed.;Porto Alegre, Artmed, 2001.

MARIANI, M. A. ,GERHARDT, G. J. L., SILVA, S. de A.;MARTINOTTO, A. L., ECHEVERRIGARAY, S. ; Utilização de *Grids* Computacionais para a Paralelização de um Algoritmo de Aprendizado de Máquina; XVII Encontro de Jovens Pesquisadores da UCS; 2009

MOUNT, David W. Bioinformatics : Sequence and Genome Analysis. New York: Cold Spring Harbor Laboratory, 2000.

NELSON,D. L.; COX, M. M.; Lehninger Princípios de Bioquímica.; 3.ed.; São Paulo, Sarvier, 2002.

RegulonDB. Disponvel em: <http://regulondb.ccg.unam.mx>. Acesso em: outubro. 2009.

RUSSELL, S.;NORVIG, P.;Inteligencia Artificial; Rio de Janeiro. Elsevier. 2004.

SANTOS,L. dos;Caracterização Computacional de Padrões Estruturais em Sequências de DNA Relacionadas a Processos em Redes Metabólicas;2009;127p.; Dissertação (Trabalho de Conclusão de Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos.

SEEFELD, K.; R For Bioinformatics;OREILLY & ASSOC;2005

SHANNON, C.; WEAVER, W. The Mathematical theory of communication. 5 ed. Urbana/Chicago, Illinois State University Press, 1963.

SILVA, S. de A.;Redes Neurais Artificiais Aplicadas na Caracterização e Predição de Regiões Promotoras;2006;144 p.; Dissertação (Trabalho de Conclusão de Mestrado em Computação Aplicada) – Universidade do Vale dos Sinos (UNISINOS), São Leopoldo.

TANENBAUM, A. S.; WOODHULL, A. S.; Sistemas Operacionais: Projeto e Implementação;2 ed.; Porto Alegre, Bookman, 2000.

TANENBAUM, A. S.;Organização Estruturada de Computadores; Rio de Janeiro, RJ, Livros Técnicos e Científicos Editora, 2007.

VARGAS, P. K.; BARRETO, M. E. Gerência de Recursos em Ambientes de Grade. Canoas, RS: ERAD 2005 - 5a Escola Regional de Alto Desempenho, SBC, Unilasalle, UFPel, UCPel, UCS, 2005.

VOET, D.; VOET J. G.; Bioquímica;3.ed;Porto Alegre, Artmed,2006.

WHEELER, David L.; BARRET, Tanya; BENSON, Dennis A.; BRYANT, Stephen H.; Canese, Kathi; CHETVERNIN, Vyacheslav; CHURCH, Deanna M.; DICUCCIO, Michael; EDGAR, Ron; FEDERHEN, Scott; GEER, Lewis Y.; HELMBERG, Wolfgang; KAPUSTIN, Yuri; KENTON, David L.; KHOVAYKO, Oleg; LIPMAN, David J.; MADDEN, Thomas L.; MAGLOTT, James Ostell; PRUITT, Kim D.; SCHULER, Gregory D.; SCHRIML, Lynn M.; SEQUEIRA, Edwin; SHERRY, Stephen T.; SIROTKIN, Karl; SOUVOROV, Alexandre; STARCHENKO, Grigory; SUZEK, Tubga O.; TATUSOV, Roman; TATUSOVA, Tatiana A.; WAGNER, Lukas; YASCHENKO, Eugene. Database resources of the National Center for Biotechnology Information. Nucleic Acids Research**,** v. 34, 2006.

ANEXO A

A.1 Pseudo algoritmo do método de cálculo da entropia

```
1 PROGRAMA_ENTROPIA
2 DECLARA variaveis
3 ENQUANTO(! FIM_DE_ARQUIVO_LE(ENTRADA,ene))
4 {
5   SALVA(ene -> genome)
6   cont2 <- cont2+1
7   cont1 <- cont1+1;
8   SE(cont1==cont2)
9   {
10    SE(genome[cont2]==0 OR genome[cont2]==2)
11      gc <- gc+1;
12    SE(cont2 mod 3==0)
13    {
14      l <- 0;
15      PARA(j <- 0; j<4; j <- j+1)
16        PARA(k <- 0; k<4; k <- k+1)
17          PARA(m <- 0; m<4; m <- m+1)
18          {
19            SE(genome[cont2]==j AND genome[cont2-1]==k AND genome[cont2-2]==m)
20              cont[l] <- cont[l]+1;
21            l <- l+1;
22          }
23    }
24    SE(cont2==n_wind)
25    {
26      S <- 0.;
27      PARA(i <- 0; i<64; i <- i+1)
28        SE(cont[i]!=0)
29          S <- S-(float)log((float)3.*cont[i]/n_wind)*((float)3.*cont[i]/n_wind);
30      S <- S/log(64.);
31      x <- (float)gc/n_wind;
32      y <- 0.52299+2.49299*x-4.93538*x*x+4.88965*x*x*x-2.44855*x*x*x*x;
33      ESCREVE_NO_ARQUIVO((float)gc/n_wind, S-y)
34      PARA(i <- 0; i<64;i <- i+1)
35        cont[i] <- 0;
36      cont2 <- 0;
37      gc <- 0;
38      norm <- 0;
39    }
40  }
41  cont1 <- cont2;
42 }
```

A.2 Pseudo algoritmo de cálculo da periodicidades

```
1 ENQUANTO(NOT FINAL_DE_ARQUIVO_LE(ENTRADA,ene) AND cont<=n_bases )
2 {
3   SALVA(ene, genome)
4   cont <- cont+1
5   cont2 <- cont2+1
```

```
6   SE(cont==n_bases)
7   {
8     cont3 <- cont3+1
9     PARA(k <- 0; k<=n_bases; k++)
10      PARA(cont <- 0; cont<=n_bases; cont <- cont+1)
11        SE(genome[cont]==genome[(cont+k+1) mod n_bases])
12          sss[k]=sss[k]+1;
13    ymax=0;
14    PARA(cont <- 0; cont<n_bases-1; cont <- cont+1)
15    {
16      y[cont] <- sss[cont]*t;
17      ymax <- ymax+y[cont];
18    }
19    ABRE_ARQUIVO(TESTE.SER)
20    PARA(i=0;i<n_bases;i++)
21    {
22      y2[i]=y[i]-(ymax/n_bases);
23      ESCREVE_NO_ARQUIVO(TESTE.SER,y2[i])
24    }
25    FECHA_ARQUIVO(TESTE.SER)
26    CHAMA_FOURIER(TESTE.SPE,TESTE.SER)
27    ABRE_ARQUIVO(TESTE.SPE)
28    i <- 0;
29    ENQUANTO(NOT FIM_DE_ARQUIVO_LE(TESTE.SPE, c1[i],c2[i]))
30    {
31      spe[i] <- spe[i]+c2[i];
32      i <- i+1;
33    }
34    FECHA_ARQUIVO(TESTE.SPE)
35    PARA(k <- 0; k<=n_bases; k <- k+1)
36      sss[k] <- 0;
37    cont <- 0;
38  }
39 }
40 FECHA_ARQUIVO(ENTRADA)
41 PARA(i <- 0; i<n_bases/2.; i <- i+1)
42   ESCREVE_NA_SAIDA(c1[i],spe[i]/cont3)
```

A.3 Pseudo algoritmo de cálculo do coeficiente de clusterização

```
1 PROGRAMA_CRIA_GRAFO
2 m2=6;
3 cont2=0;
4 ENQUANTO(cont2<m2 )
5 {
6   SE(! FIM_DE_ARQUIVO(ENTRADA))
7   {
8     LE_ARQUIVO(ENTRADA,genome)
9   }
10  cont2 <- cont2+1
11  SE(cont2==m2)
12  {
13    l <- -1
14    PARA(j <- 0; j<4; j <- j+1)
15      PARA(k <- 0; k<4; k <- k+1)
16        PARA(m <- 0; m<4; m<- m+1)
17        {
18           l <- l+1;
19           SE(genome[1]==j AND genome[2]==k AND genome[3]==m)
20             cont[l] <- cont[l]+1;
21         }
22    l <- -1;
23    PARA(j <- 0; j<4; j <- j+1)
24      PARA(k <- 0; k<4; k <- k+1)
25        PARA(m <- 0; m<4; m <- m+1)
26        {
27          l <- l+1;
28          SE(genome[4]==j AND genome[5]==k AND genome[6]==m)
29            contf[l] <- cont[l]+1;
```

```
30        }
31    PARA(l <- 0; l<64; l <- l+1)
32      SE(cont[l]!=0)
33        GRAVA_ARQUIVO(SAIDA, l)
34
35    PARA(l <- 0; l<64; l <- l+1 )
36      SE(contf[l]!=0)
37        GRAVA_ARQUIVO(SAIDA, l)
38
39    PARA(j <- 0; j<64; j <- j+1)
40    {
41      cont[j] <- 0;
42      contf[j] <- 0;
43    }
44    cont2 <- 0;
45  }
46 }
```

A saída do pseudo-algoritmo acima serve como entrada no pseudo-algoritmo abaixo.

```
1  PROGRAM_CALCULA_COEFICIENTE_DISPERSAO
2  SE(n_ligad>500)
3    factor <- (0.3/1900.)*(float)n_ligad+0.455;
4  SE(n_ligad<=500)
5    factor <- (0.3/1300.)*(float)n_ligad+0.415;
6  SE(n_ligad<=150)
7    factor <- factor/2.;
8
9  kk <- 1;
10 nmax <- atoi(argv[3]);
11 nmax <- pow(4,nmax);
12 SE(nmax>pow(4,4))
13 {
14   IMPRIME("... n_uplas too big, greather than 4.\n");
15   exit(0);
16 }
17 ENQUANTO(kk<=2)
18 {
19   PARA(ii <- 0; ii<=nmax; ii <- ii+1)
20   {
21     i11[ii] <- i22[ii] <- icl[ii] <- idg[ii] <- 0;
22     coef[ii] <- 0.;
23     PARA(jj <- 0;jj<=nmax;jj <- jj+1)
24        mat[ii][jj] <- 0;
25   }
26   i <- 1;
27   ENQUANTO(i <= n_ligad)
28   {
29     SE(! FIM_DE_ARQUIVO(ENTRADA)
30     {
31       LE_ARQUIVO(ENTRADA,i11,i22)
32       exit(0);
33     }
34     mat[i11[i]+1][i22[i]+1] <- 1;
35     mat[i22[i]+1][i11[i]+1] <- 1;
36     i <- i+1;
37    }
38 nump <- i-1;
39 PARA(i <- 1; i<=nmax; i <- i+1)
40   PARA(j <- 1; j<=nmax; j <- j+1)
41     idg[i] <- idg[i]+mat[i][j];
42
43 PARA(i <- 1; i<=nmax; i <- i+1)
44   PARA(j <- 1; j<=nmax; j <- j+1)
45     PARA(i1 <- 1; i1<=nmax; i1 <- i1+1)
46       PARA(i2 <- i1+1; i2<=nmax; i2 <- i2+1)
47         SE(mat[i][i2]==1)
48           SE(j==i1)
49             icl[i] <- icl[i]+mat[i][j]*mat[i1][i2];
50 zzz <- 0;
51 PARA(i <- 1; i<=nmax; i <- i+1)
52 {
```

```
53   SE(idg[i]==1 OR icl[i]==0 OR idg[i]==0)
54   {
55     coef[i] <- 0;
56   }
57   SENAO
58   {
59     coef[i] <- 2.*icl[i]/(idg[i]*(idg[i]-1));
60     zzz <- zzz+1;
61   }
62 }
63 xcl <- 0.;
64 PARA(i <- 1; i<=nmax; i <- i+1)
65   xcl <- xcl+coef[i];
66 GRAVA_NO_ARQUIVO(SAIDA,(float) factor*xcl/nmax/(1.*nump/nmax/nmax)))
67 }
```

A.4 Pseudo algoritmo de cálculo da dispersão do coeficiente de clusterização

```
1 FUNCAO l1(int l, float gc, float c)
2 {
3
4   SE(l==250)
5   {
6     m=0.76853+44.83098*gc-459.55064*gc*gc+2426.76792*gc*gc*gc-
        7775.6105*gc*gc*gc*gc+15067.61486*gc*gc*gc*gc*gc-
        17010.7263*gc*gc*gc*gc*gc*gc+10275.46666*gc*gc*gc*gc*gc*gc*gc-
        2568.79288*gc*gc*gc*gc*gc*gc*gc*gc;
7     s=0.24366+1.52452*gc-19.99541*gc*gc+145.31602*gc*gc*gc-
        605.11222*gc*gc*gc*gc+1369.
        51997*gc*gc*gc*gc*gc-
        1671.94889*gc*gc*gc*gc*gc*gc+1039.99425*gc*gc*gc*gc*gc*gc*gc-
        259.29257*gc*gc*gc*gc*gc*gc*gc*gc;
8   }
9   SE(l==500)
10  {
11    m=1.17226+5.72497*gc+35.48471*gc*gc-429.47988*gc*gc*gc+1395.74283*gc*gc*gc*gc-
        2290.33891*gc*gc*gc*gc*gc+2211.54849*gc*gc*gc*gc*gc*gc-
        1234.31371*gc*gc*gc*gc*gc*gc*gc+305.60173*gc*gc*gc*gc*gc*gc*gc*gc;
12    s=0.42493-10.15892*gc+133.83398*gc*gc-819.87582*gc*gc*gc+2709.0574*gc*gc*gc*gc-
        5183.99852*gc*gc*gc*gc*gc+5762.73214*gc*gc*gc*gc*gc*gc-
        3452.10131*gc*gc*gc*gc*gc*gc*gc+860.50603*gc*gc*gc*gc*gc*gc*gc*gc;
13  }
14  SE(l==1000)
15  {
16    m=0.71824+6.41216*gc+29.02373*gc*gc-452.55345*gc*gc*gc+1805.98511*gc*gc*gc*gc-
        3648.44518*gc*gc*gc*gc*gc+4139.0364*gc*gc*gc*gc*gc*gc-
        2506.4119*gc*gc*gc*gc*gc*gc*gc+626.94242*gc*gc*gc*gc*gc*gc*gc*gc;
17    s=0.04189+4.03678*gc-57.10141*gc*gc+364.01026*gc*gc*gc-
        1263.56363*gc*gc*gc*gc+2516.63032*gc*gc*gc*gc*gc-
        2862.8493*gc*gc*gc*gc*gc*gc+1727.84817*gc*gc*gc*gc*gc*gc*gc-
        428.98421*gc*gc*gc*gc*gc*gc*gc*gc;
18  }
19  SE(l==2000)
20  {
21    m=0.38651+10.68037*gc-58.35254*gc*gc+139.05402*gc*gc*gc-119.0997*gc*gc*gc*gc-
        102.26208*gc*gc*gc*gc*gc+294.57506*gc*gc*gc*gc*gc*gc-
        214.42249*gc*gc*gc*gc*gc*gc*gc+49.78478*gc*gc*gc*gc*gc*gc*gc*gc;
22    s=0.51039-7.83025*gc+56.18942*gc*gc-219.89799*gc*gc*gc+519.21474*gc*gc*gc*gc-
        775.89902*gc*gc*gc*gc*gc+735.34146*gc*gc*gc*gc*gc*gc-
        410.84187*gc*gc*gc*gc*gc*gc*gc+103.75821*gc*gc*gc*gc*gc*gc*gc*gc;
23  }
24  RETORNA (c-m)/s;
25 }
26
27 PROGRAM_CALCULO_DISPERSAO
28 {
29   l=atoi(argv[1]);
30   j=0;
31   i=0;
```

```
32   dmono=0;
33   gc1=0;
34   ENQUANTO(! FIM_DE_ARQUIVO)
35   {
36     LE_ARQUIVO(ENTRADA,gc,c)
37     i -> i+1;
38     dmono -> dmono+l1(l,gc,c);
39     gc1 -> gc1+gc;
40   }
41   GRAVA_ARQUIVO(SAIDA,gc1/i,dmono/i)
42 }
```

A.5 Pseudo algoritmo do método de determinação de promotores através de uma rede neural

```
1 argumentos<-commandArgs()
2 oculto<-as.numeric(argumentos[[4]])
3 i<-as.numeric(argumentos[[5]])
4 iteracoes<-as.numeric(argumentos[[6]])
5 lista<-as.numeric(argumentos[[7]])
6
7 library(nnet)
8
9 treino <- list()
10 treino[[1]] <- CARREGA_ARQUIVO(TREINO1)
11 treino[[2]] <- CARREGA_ARQUIVO(TREINO2)
12 treino[[3]] <- CARREGA_ARQUIVO(TREINO3)
13 treino[[4]] <- CARREGA_ARQUIVO(TREINO4)
14 treino[[5]] <- CARREGA_ARQUIVO(TREINO5)
15 treino[[6]] <- CARREGA_ARQUIVO(TREINO6)
16 treino[[7]] <- CARREGA_ARQUIVO(TREINO7)
17 treino[[8]] <- CARREGA_ARQUIVO(TREINO8)
18 treino[[9]] <- CARREGA_ARQUIVO(TREINO9)
19 treino[[10]] <- CARREGA_ARQUIVO(TREINO10)
20
21 teste<-list()
22 teste[[1]] <- CARREGA_ARQUIVO(TESTE1)
23 teste[[2]] <- CARREGA_ARQUIVO(TESTE2)
24 teste[[3]] <- CARREGA_ARQUIVO(TESTE3)
25 teste[[4]] <- CARREGA_ARQUIVO(TESTE4)
26 teste[[5]] <- CARREGA_ARQUIVO(TESTE5)
27 teste[[6]] <- CARREGA_ARQUIVO(TESTE6)
28 teste[[7]] <- CARREGA_ARQUIVO(TESTE7)
29 teste[[8]] <- CARREGA_ARQUIVO(TESTE8)
30 teste[[9]] <- CARREGA_ARQUIVO(TESTE9)
31 teste[[10]] <- CARREGA_ARQUIVO(TESTE10)
32
33 r <- list()
34 Lst <- list()
35
36 Lst[[1]] <- nnet(x=treino[[1]][ ,1:324],y=treino[[1]]
[ 325],size=oculto,maxit=0,MaxNWts=100000)
37 Lst[[2]] <- nnet(x=treino[[2]][ ,1:324],y=treino[[2]][ ,
325],size=oculto,maxit=0,MaxNWts=100000)
38 Lst[[3]] <- nnet(x=treino[[3]][ ,1:324],y=treino[[3]][ ,
325],size=oculto,maxit=0,MaxNWts=100000)
39 Lst[[4]] <- nnet(x=treino[[4]][ ,1:324],y=treino[[4]][ ,
325],size=oculto,maxit=0,MaxNWts=100000)
40 Lst[[5]] <- nnet(x=treino[[5]][ ,1:324],y=treino[[5]][ ,
325],size=oculto,maxit=0,MaxNWts=100000)
41 Lst[[6]] <- nnet(x=treino[[6]][ ,1:324],y=treino[[6]][ ,
325],size=oculto,maxit=0,MaxNWts=100000)
42 Lst[[7]] <- nnet(x=treino[[7]][ ,1:324],y=treino[[7]][ ,
325],size=oculto,maxit=0,MaxNWts=100000)
43 Lst[[8]] <- nnet(x=treino[[8]][ ,1:324],y=treino[[8]][ ,
325],size=oculto,maxit=0,MaxNWts=100000)
44 Lst[[9]] <- nnet(x=treino[[9]][ ,1:324],y=treino[[9]][ ,
325],size=oculto,maxit=0,MaxNWts=100000)
45 Lst[[10]] <- nnet(x=treino[[10]][ ,1:324],y=treino[[10]][ ,
```

```
325],size=oculto,maxit=0,MaxNWts=100000)
46
47
48 r[[i]] <- nnet(x=treino[[i]][ ,1:324],y=treino[[i]][ ,325], Wts=Lst[[lista]]
$wts,size=oculto, maxit=iteracoes, MaxNWts=10000,linout=TRUE)
49 erro1<- predict(r[[i]],teste[[i]][ ,1:324])-teste[[i]][,325]
50 write(x=erro1,
file=(paste("rede",i,iteracoes,"hidden",oculto,"list",lista,"Erroteste.txt")),ncolumn
s=1)
51 sum(erro1)
52 sum(erro1*erro1)
53 sum(erro1*erro1)/28
54 raiztes1 <- sqrt(sum(erro1*erro1)/28)
55 write(x=raiztes1, file=(paste("rede",i,iteracoes,"hidden",oculto,"RMSteste.txt")),
ncolumns=1,append=TRUE)
56 errot1<- predict(r[[i]],treino[[i]][ ,1:324])-treino[[i]][,325]
57 write (x=errot1,
file=(paste("rede",i,iteracoes,"hidden",oculto,"list",lista,"Errotreino.txt")),ncolum
ns=1)
58 sum(errot1)
59 sum(errot1*errot1)
60 sum(errot1*errot1)/251
61 raiz1tre <- sqrt(sum(errot1*errot1)/251)
62 write(x=raiz1tre,
file=(paste("rede",i,iteracoes,"hidden",oculto,"RMStreino.txt")),ncolumns=1,append=TR
UE)
```

Anexo B

B.1 Resumo apresentado no Jovem Pesquisador 2009

Utilização de Grids para Paralelização do Processamento de Redes Neurais Artificiais

Mauricio Adami Mariani (Voluntário), Scheila de Avila e Silva, André Martinotto, Gunther Johannes L. Gerhardt, Sergio Echeverrigaray (orientador) - wushubr@gmail.com

Os avanços na genômica possibilitaram o aumento da quantidade de informação genética dos seres vivos. Para realizar a análise desse grande volume de dados, a utilização de computação de alto desempenho se torna necessária. Uma alternativa para a sua realização é o uso de *grids* computacionais. Esses se baseiam na utilização de computadores não dedicados para a obtenção de uma plataforma de alto desempenho, ou seja, visa a utilização de ciclos ociosos de computadores para a solução de problemas que necessitam de um alto poder de processamento. O presente trabalho teve como objetivo paralelizar o método de Redes Neurais Artificiais aplicadas na predição de sequências promotoras de organismos procariotos em um ambiente computacional de alto desempenho, utilizando *grids*. Os *grids* foram escolhidos porque dividem o problema a ser resolvido em partes independentes. A paralelização consistiu em particionar a metodologia de *tenfold cross-validation* e as repetições necessárias para validar a técnica de Redes Neurais Artificiais estatisticamente. Assim, as 1.600 simulações necessárias para a análise dos dados que consumiam 29 horas de processamento em um único computador passaram a serem realizadas com um tempo de processamento de 39 minutos.
Palavras-chave: computação de alto-desempenho, bioinformática, *grids*.

Apoio: UCS.
XVII Encontro de Jovens Pesquisadores – Setembro de 2009
Pró-Reitoria de Pós-Graduação e Pesquisa
Universidade de Caxias do Sul