Técnicas de Paralelismo Aplicadas à Execução de Fluxos de Programas de Bioinformática

Carlos Eduardo Costa Vieira
e-mail: cadu@inf.puc-rio.br

Luiz Fernando Bessa Seibel
e-mail: seibel@inf.puc-rio.br

PUC-RioInf.MCC23/04 Junho, 2004

Abstract: One of the most important tasks of the genome projects is called process of annotation, which consists in the analysis and interpretation of experimental data in order to derive biological knowledge from the data. Many systems were created to aid the researchers in the annotation process, such as BioNotes. This tool is currently used to annotate the genome of the bacteria *Gluconacetobacter diazotrophicus* and supports three types of annotations: imported, originating from external data source; automatics, generated by execution of biological analysis programs, and manual, created by the biologists. This paper refers to the automatic annotations, that is essential for the researchers but which spend a lot of time to be available in the BioNotes system. This time can be reduced if parallel computing methods were used in the execution of the workflow which generates automatic annotations. The objective of this paper is to discuss these parallel computing methods and to demonstrate the viability and importance of the parallelism methods in the bioinformatics area.

Keywords: annotation, BioNotes, workflow, parallel computing, distributed environment

Resumo: Uma das principais tarefas dos projetos genoma é o chamado processo de anotação, que consiste em analisar e interpretar os dados experimentais com o objetivo de obter conhecimento biológico sobre os mesmos. Diversas ferramentas surgiram para auxiliar os pesquisadores nessa tarefa, dentre elas o BioNotes. Esse ambiente vem sendo utilizado no processo de anotação do genoma da bactéria *Gluconacetobacter diazotrophicus* e possibilita que sejam feitos três tipos de anotações: importadas, provenientes de fontes de dados externas; automáticas, provenientes da execução de aplicativos da biologia e manuais, feitas pelos biólogos. Este estudo refere-se às anotações automáticas, que são essenciais para os pesquisadores, porém consomem uma grande quantidade de tempo para estarem disponíveis. Este tempo pode ser reduzido se forem aplicadas técnicas de computação paralela no fluxo de trabalho compostos pelos diversos aplicativos da biologia. Atualmente, a computação paralela em um ambiente distribuído é uma alternativa bastante viável para aplicações científicas e comerciais. Assim, este trabalho tem como principal objetivo propor a paralelização do fluxo de execução dos programas aplicativos da bactéria *Gluconacetobacter diazotrophicus* em um ambiente de memória distribuída, com o intuito de demonstrar a viabilidade e a importância da aplicação de técnicas de paralelismo na área de bioinformática.

Palavras-chave: anotação, Bionotes, fluxo de execução de aplicativos, computação paralela, ambiente distribuído

Sumário

1 – Introdução

Um dos objetivos dos projetos genoma é a completa elucidação dos pares de bases *(Adenina, Timina, Citosina, Guanina)* que compõem os cromossomos de uma determinada espécie, isto é, da seqüência de *DNA* (Ácido Desoxirribonucléico) primária do organismo. Na década passada, vários centros de pesquisa sequenciaram o genoma completo de diversos organismos, iniciando com os genomas dos micróbios e continuando com espécies eucarióticas tais como *Saccharomyces cerevesiae* (levedura), *Caenorhabditis elegans* (verme), *Drosophila melanogaster* (mosca da fruta) e *Arabidopsis thaliana* (erva daninha da mustarda), culminando mais recentemente com o anúncio dos grupos públicos e privados de versões do genoma humano. Além disso, diversos outros projetos genoma estão a caminho, incluindo os genomas de outros mamíferos como o do rato e dos primatas não-humanos [S01]. O volume de dados gerados no processo de seqüenciamento de um genoma é enorme.

Os projetos genoma estão inseridos em uma área recente de pesquisa denominada Bioinformática que tem por fim a pesquisa de modelos, técnicas e ferramentas computacionais que auxiliem o entendimento dos fenômenos biológicos. [SL00].

Em paralelo com o processo de seqüenciamento de um genoma, pode-se iniciar o chamado processo de anotação, cujo objetivo é extrair informações significativas das seqüências de *DNA*, adicionando camadas de análise e interpretação. Anotações incluem a identificação de genes, tRNAs (Ácido Ribonucléico de transferência), rRNAs (Ácido Ribonucléico ribossômico), elementos repetidos, entre outros sítios relevantes, nas seqüências do *DNA*, a determinação das funções dos genes encontrados, a descoberta de regiões do *DNA* que estão relacionadas ao mecanismo de controle e regulação dos genes, etc.

No passado, o processo de anotação era feito exclusivamente de forma manual [H97], [D01]. Porém, a taxa de seqüenciamento tem aumentado de forma exponencial, provocando a necessidade cada vez maior do uso de ferramentas que automatizem ao máximo o processo de anotação. Contudo, vale salientar que a anotação manual continua extremamente valiosa [H97]. Apesar da evolução dos algoritmos, métodos de previsão dos genes ainda não são totalmente confiáveis, necessitando-se da intervenção humana [S01]. Um outro problema extremamente delicado e que requer a intervenção

humana é o da propagação de erros de seqüenciamento em banco de dados públicos de nucleotídeos e de proteínas, causando anotações errôneas e atrapalhando esforços de anotações posteriores [FAH+01]. Desta maneira, a análise do genoma de um organismo produzida por qualquer ferramenta automática deveria ser considerada como uma primeira aproximação. Melhoramentos adicionais na qualidade dos dados ficam a cargo de biólogos e especialistas da área.

Assim, devido ao grande volume de dados e da necessidade de automatizar um grande número de operações, uma variedade de sistemas rapidamente foram desenvolvidos para auxiliar o processo de anotação, incluindo *GeneQuiz* [SSC+94], *BCM Search Launcher* [SWW+96], *Genotator* [H97], *EDITtoTrEMBL* [MLF+99], *PEDANT* [FAH+01], *RiceGAAS* [SNN+02], entre outros. À partir de um estudo realizado em diversos sistemas de anotações existentes [LSC03][1] e do levantamento realizado com os pesquisadores da área sobre as características importantes que um sistema de anotação deve possuir [LSC03][2], surgiu o BioNotes.

O BioNotes [LSC03][2] foi desenvolvido na *Pontifícia Universidade Católica do Rio de Janeiro* e atualmente vem sendo usado para auxiliar o processo de anotação de três organismos: *Trypanosoma cruzi* que está sendo anotado pelo Departamento de Bioquímica e Biologia Molecular do Instituto Oswaldo Cruz [DB04], *Gluconacetobacter diazotrophicus*, que está sendo anotado pelo projeto RioGene [R04] e *Rhodnius prolixus*, que está sendo anotado pelo Departamento de Bioquímica da *Universidade Federal do Rio de Janeiro* (UFRJ). Os exemplos de fluxos e de aplicativos utilizados neste trabalho foram extraídos do BioNotes aplicado ao processo de seqüenciamento da bactéria *Gluconacetobacter diazotrophicus*. Por questões de simplicidade, de agora em diante, esse organismo será denominado *Glucona*.

O BioNotes é um sistema distribuído de anotações biológicas utilizado pela comunidade de biologia molecular para facilitar a análise e interpretação de seus dados [LSC03][1]. Suporta três tipos de anotações [LSC03][2]: importadas, provenientes de fontes de dados externas como o GenBank [BKL+03] e o PIR [WHA+02]; automáticas, obtidas através da execução de diversos programas de análise como o Phred [EHW+98] [EG98], Phrap [G03] e BLAST *(Basic Local Alignment Search Tool)* [AGM+90] e manuais, realizadas pelos biólogos a partir de suas próprias observações e dos resultados provenientes dos dois tipos de anotações anteriores. Todas essas anotações são armazenadas no *data*

warehouse do BioNotes e podem ser consultadas via *World Wide Web* de diversas maneiras [LSC03][1].

As anotações automáticas do BioNotes, geradas através da execução seqüencial de diversos aplicativos externos (*workflow* seqüencial) e posterior inserção no *data warehouse*, é uma tarefa que consome uma grande quantidade de tempo (algumas semanas) e tem-se mostrado um gargalo para o sistema como um todo. Contudo, ela é uma tarefa essencial porque o pré-processamento e o armazenamento das anotações automáticas no *data warehouse* permite que os pesquisadores tenham acesso mais rápido à estas anotações.

Com o intuito de melhorar o desempenho do processo mencionado no parágrafo anterior, pode-se fazer uso de máquinas multiprocessadas e de paralelismo. Com os avanços nas tecnologias de microcomputadores e de redes, o emprego de máquinas com arquitetura de memória distribuída tem se tornado viável apresentando uma relação de custo x desempenho bastante atrativa [C02]. Assim, um dos objetivos deste trabalho é o de apresentar uma proposta de paralelização do fluxo de execução dos programas utilizados para gerar as anotações automáticas da *Glucona*, que será denominado aqui de *workflow* paralelo. Essa proposta do *workflow* paralelo estará baseada em estudos anteriores realizados com a ferramenta BLAST em uma máquina paralela com arquitetura de memória distribuída, mais especificamente, em um *cluster* de *PCs* [C02], [CL03].

Esse trabalho está organizado como se segue: a Seção 2 apresenta um resumo do trabalho feito no BioNotes aplicado à *Glucona*, mostrando a importância da bactéria em diversas culturas agrícolas, as principais características do BioNotes e detalhando o fluxo de execução seqüencial dos programas aplicativos; a Seção 3 aborda a questão da aplicação de técnicas de paralelismo na pesquisa em biologia molecular, iniciando-se com alguns conceitos em computação paralela importantes para o trabalho, em seguida descrevendo a utilização da ferramenta BLAST em um *cluster* de *PCs* e concluindo com quatro propostas de um *workflow* paralelo para os programas aplicativos da *Glucona* e uma comparação entre as mesmas; e por fim, a Seção 4 mostra as considerações finais do presente estudo e as propostas de trabalhos futuros.

2 – Workflow Seqüencial da Glucona

2.1 – O Organismo Gluconacetobacter diazotrophicus

A bactéria *Gluconacetobacter diazotrophicus* é responsável pela fixação biológica do gás nitrogênio (N_2), retirando-o do ar e do solo e transformando-o em um composto que estimula o crescimento da planta. Essa bactéria está presente em diversas culturas de grande importância agrícola, como a cana-de-açúcar, o café, a banana e a batata doce. A *Glucona* não vive no solo, é uma endófita obrigatória, ou seja, tem que crescer no interior do vegetal para poder sobreviver e não causa nenhum tipo de doença à planta. O que se pretende com o seqüenciamento da *Glucona* é aprimorar o rendimento metabólico da bactéria, ampliando sua capacidade de fixação biológica de nitrogênio. Estudos preliminares estimam que, reduzindo em 30% a quantidade de fertilizantes nitrogenados aplicados em toda a área cultivada com cana-de-açúcar no país - cerca de 4 milhões de hectares na safra 2001/2002 - seria possível economizar em torno de R$ 100 milhões por ano. A bactéria diminui também indiretamente a poluição, já que os adubos podem contaminar rios e lençóis freáticos [FP03], [UF03].

2.2 – BioNotes Aplicado à Glucona

O BioNotes é a ferramenta que está sendo utilizada para auxiliar os pesquisadores e cientistas no processo de anotação da bactéria *Glucona*. Assim, para este grupo de pesquisa, uma área de trabalho foi personalizada e os usuários pertencentes a essa comunidade ou grupo de pesquisa foram devidamente cadastrados. No BioNotes, perfis de usuários podem ser criados, limitando, por exemplo, o acesso a algumas funcionalidades do sistema. Usuários não cadastrados não possuem acesso aos dados armazenados no BioNotes [LSC03][1] .

O BioNotes foi desenvolvido utilizando a infraestrutura do Bio-AXS [SL01], um *data warehouse* biológico que integra dados e aplicativos de várias fontes externas [LSC03][2]. Por questões de portabilidade e por ser uma linguagem orientada a objetos, fato que permite a reutilização do código, o sistema foi implementado utilizando a linguagem de programação *Java 1.4.0*. A interface *Web*, desenvolvida através de *HTML (HyperText Markup Language)* e *JSP (Java Server Pages)*, permite a distribuição das anotações, podendo ser compartilhada por diversos usuários e acessada de qualquer computador

conectado à *Internet* [LSC03][1]. A ferramenta utiliza também um sistema gerenciador de banco de dados comercial, denominado aqui pela sigla SGBD [LSC03][2].

Os tipos de anotações suportados pelo BioNotes são: importadas, automáticas e manuais [LSC03][2]. As anotações são primeiramente analisadas, transformadas para um formato de dados semi-estruturado (*XML - eXtensible Markup Language)* e armazenadas no *data warehouse* [LSC03][1]. O padrão *XML* foi adotado porque é baseado em idéias simples e está sendo utilizado cada vez mais pela comunidade de biólogos. Por exemplo, algumas fontes de dados externas como o GenBank e o PIR já possuem os seus dados representados em *XML* e alguns aplicativos como o BLAST já disponibilizam a sua saída em formato *XML* [LSC03][1].

Uma característica interessante do BioNotes é a capacidade de manter diversas versões das anotações, o que é especialmente útil para anotar dados de genomas com seqüenciamento ainda incompleto, fato que acontece com a *Glucona* [LSC03][1]. À medida que o seqüenciamento avança, novos *contigs* e *reads* continuam a ser gerados. Assim, de acordo com a necessidade dos pesquisadores da *Glucona*, os novos dados são inseridos no *data warehouse*, os programas aplicativos são novamente executados e seus resultados armazenados, configurando-se assim uma nova versão de dados do organismo no BioNotes. As anotações feitas em uma versão são migradas para a outra, desde que sejam feitas sobre os mesmos *contigs*. Isso possibilita que a fase de anotação seja realizada em paralelo com a fase de seqüenciamento, acelerando-se o processo de conhecimento do genoma da bactéria.

As anotações armazenadas no BioNotes podem ser consultadas de diversas maneiras [LSC03][1], [LSC03][2]:

- pode-se pesquisar por *contigs*, *ORFs (Open Reads Frames)*, *reads*, *singletons* de uma determinada versão da *Glucona*, como mostra a Figura 1;

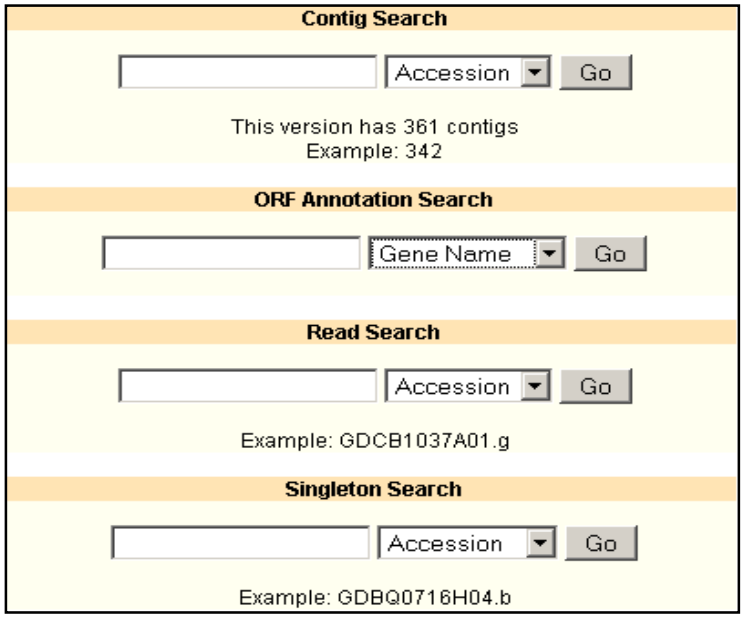

Figura 1 – Pesquisa por *contigs*, *ORFs*, *reads*, e *singletons* em uma determinada versão da Glucona

- pode-se visualizar os resultados da execução dos aplicativos através de gráficos dos padrões biológicos importantes como *ORFs*, tRNAs, terminadores, sítios de ligações de ribossomos, etc, como mostra a Figura 2*;*

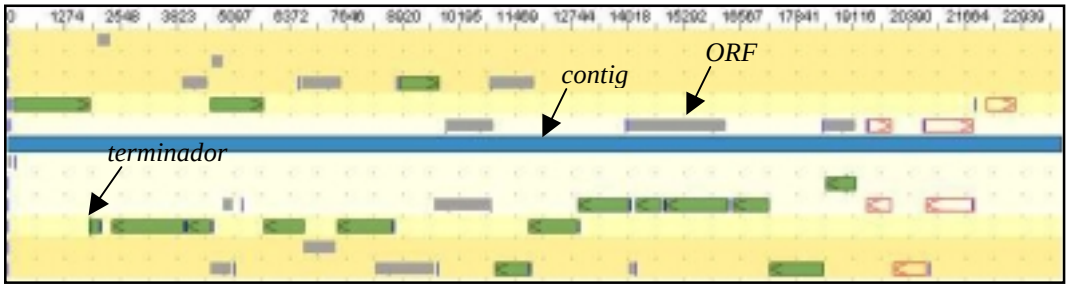

Figura 2 – Visualização de padrões biológicos no BioNotes

- pode-se obter as seqüências homólogas de uma *ORF* contra o NR (uma base de dados de aminoácidos) através da execução do BLAST;

- pode-se acessar, através de *links*, fontes de dados externas;

- os usuários podem cadastrar e atualizar (criando uma nova versão da anotação) as anotações, disponibilizando-as para toda a comunidade;

- os usuários podem também remover as suas próprias anotações.

Programas *on-line* também estão disponíveis no BioNotes como diversos tipos de BLAST, HMMTOP [TS98], [TS01] e Clustalw [THG94], dando aos usuários

flexibilidade e rapidez nas consultas. Existe disponível no BioNotes um Fórum de Discussões, o que permite uma maior interação entre os próprios usuários da *Glucona* e entre os usuários e os projetistas/desenvolvedores do BioNotes.

O BioNotes tem-se mostrado uma ferramenta bastante flexível, pois aceita em seu ambiente o estudo de diversos organismos, além de versões de um mesmo organismo em genomas ainda incompletos. A ferramenta disponibiliza a execução de aplicativos de forma *on-line* e possibilita uma melhor interação entre os pesquisadores através do Fórum de Discussões. Essa ferramenta tem dado uma grande contribuição aos projetos genoma que a utilizam.

2.3 – Descrição do Workflow Seqüencial da Glucona

Os resultados da execução dos aplicativos é responsável pela chamada anotação automática no BioNotes. De acordo com as necessidades do grupo de pesquisa da *Glucona*, o fluxo de execução dos aplicativos foi elaborado e é mostrado na Figura 3. As figuras retangulares indicam os diversos programas utilizados e as setas indicam os fluxos de entrada e de saída de cada programa. Basicamente, pode-se dividir o *workflow* seqüencial da *Glucona* em oito partes, indicadas pelos números dentro dos círculos pontilhados. Cada parte pode ser expandida, gerando o que será denominado *workflow* seqüencial do programa específico. Assim, por exemplo, expandindo-se o fluxo 2, ter-se-á o *workflow* seqüencial do *tRNAscan-SE* [LE97], expandindo-se o fluxo 3 ter-se-á o *workflow* seqüencial do *Glimmer* [DHK+99], e assim, sucessivamente. Várias dificuldades foram enfrentadas para se chegar ao *workflow* seqüencial da Glucona: os aplicativos utilizados são escritos em linguagens de programação diferentes, não estando disponíveis para sistemas operacionais e plataformas uniformes, sendo que alguns ambientes são pouco conhecidos. Alguns aplicativos estão mal documentados, isto é, não possuem uma documentação detalhada sobre os parâmetros de execução do programa e nem exemplos que facilitem o seu entendimento, possuem formatos de entrada e de saída diferentes e alguns apresentam até mesmo problemas durante a sua execução, dificultando ainda mais o seu uso.

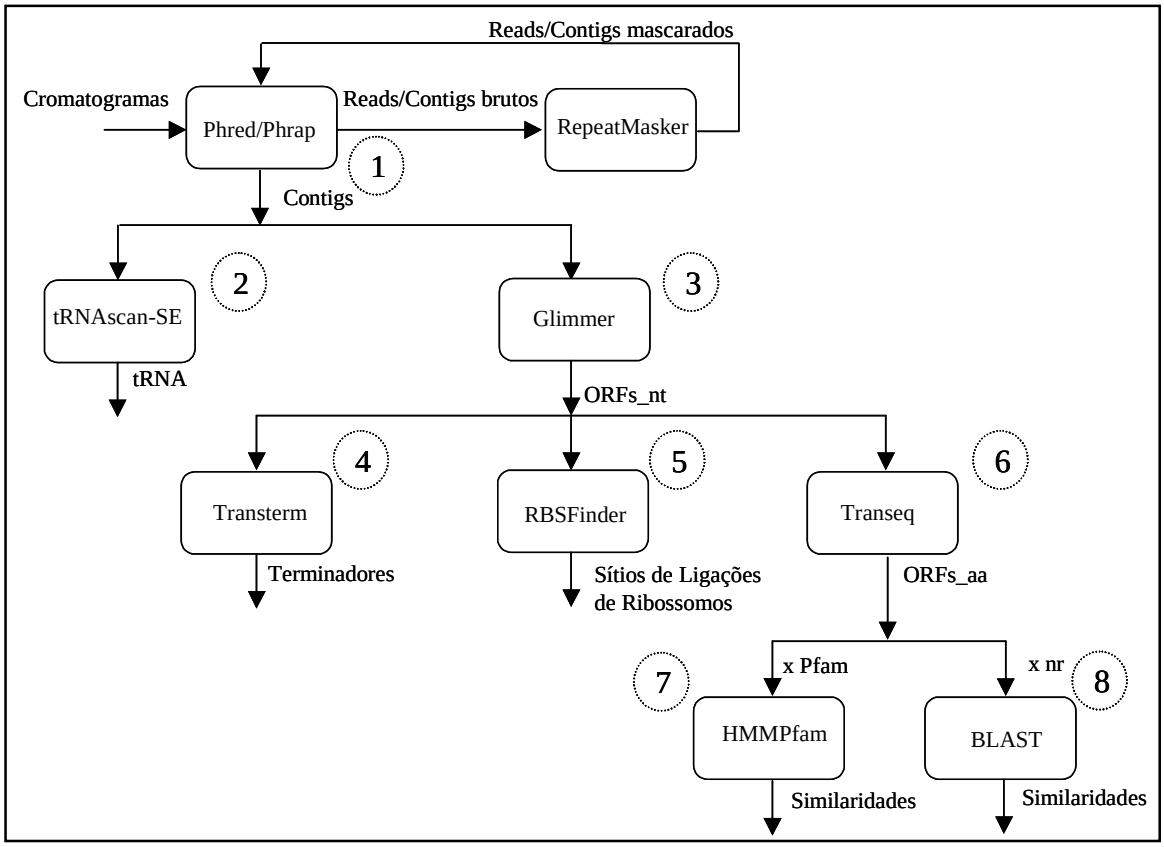

Figura 3 – Fluxo de execução dos programas no projeto da *Glucona*

O fluxo de execução dos aplicativos sobre os dados da *Glucona* inicia-se a partir dos cromatogramas seqüenciados nos laboratórios do RioGene. Primeiramente, o Phred e o Phrap são executados sobre os cromatogramas com o intuito de obter os *reads* e *contigs* brutos. Em seguida, o programa RepeatMasker [SG03] identifica e mascara repetições no genoma da *Glucona* usando um conjunto de repetições obtidas através de análises de outros organismos. Posteriormente, o Phred e o Phrap são executados sobre os *contigs* e *reads* mascarados. Esta descrição corresponde à parte 1 da Figura 3 e está sob a responsabilidade dos pesquisadores do RioGene. Contudo, essa parte poderia ser executada pelo BioNotes, sem nenhum problema.

De posse dos arquivos de saída do Phred e do Phrap, pode-se inserir no *data warehouse* do BioNotes padrões importantes como os *reads*, *contigs* e *singletons*. Antes da inserção no SGBD, um *parser* transforma os dados do formato específico de saída do Phred e do Phrap para o formato XML.

Paralelamente, tendo-se como entrada os *contigs*, pode-se executar os aplicativos tRNAscan-SE e Glimmer com o objetivo de encontrar possíveis tRNAs ou ORFs (possíveis genes), o que corresponde às partes 2 e 3 do *workflow* seqüencial da *Glucona*,

respectivamente. O *workflow* seqüencial do programa tRNAscan-SE pode ser expandido como mostra a Figura 4.

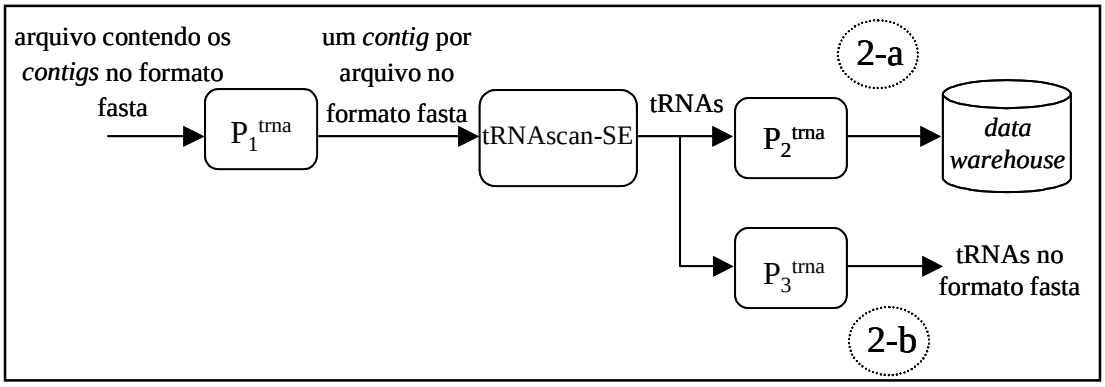

Figura 4 – *Workflow* seqüencial do tRNAscan-SE

Os seguintes passos são executados até que todos os possíveis tRNAs encontrados estejam no *data warehouse* do BioNotes (fluxo 2-a):

- o *parser* P_1^{trna} transforma o arquivo que contém todos os *n contigs* no formato fasta para *n* arquivos com um único *contig* por arquivo, onde *n* representa a quantidade de *contigs* em uma determinada versão da *Glucona*;

- para cada *contig*, executa-se o tRNAscan-SE;

- o *parser* P_2^{trna} é responsável por transformar a saída do aplicativo em XML e por inserir os tRNAs encontrados no *data warehouse*.

O formato fasta, largamente utilizado pelos biólogos, apresenta uma linha identificadora da seqüência, iniciada pelo caracter '>', seguida por outras linhas contendo a seqüência propriamente dita (Figura 5). Uma operação muitas vezes necessária é transformar os tRNAs encontrados para o formato fasta (fluxo 2-b). Assim, o *parser* P_3^{trna} lê o resultado padrão do tRNAscan-SE que contém a seqüência dos tRNAs encontrados nos *contigs*, transformando-a para o formato fasta.

```
>Contig342
CCGGAGCCTGCAGAGCTTGCTCTCTGAAGCTTNTACGAAGNATGGGCTACGCGAAGTTGCTC
TCTGGAAGTTTTGTAGAAAGCGGTGGGCTACCGCTATTGTTCTCCACACTCCTCTATAGCGG
TAGCACCGATTTTTCCCTACTGCCGGAATCGATTGACTCATCAATCGTGTCCTCGACATTCT
CACGTTGCTCTCTTCCCCTGCCCTGTCTCCCACCGATACAGAAATTTCCATTGATGATCTTG
CACTCCTCTGATTCGTTTTTCTCGCCTCCTCCGCTGTCTCCTCCTTTTCTGTCCCTACTCCC
ATTACCACCAGATTGATTGATCAATCAATCA
```

Figura 5 – Exemplo de uma seqüência no formato fasta

O *workflow* seqüencial do programa Glimmer pode ser expandido como mostra a Figura 6.

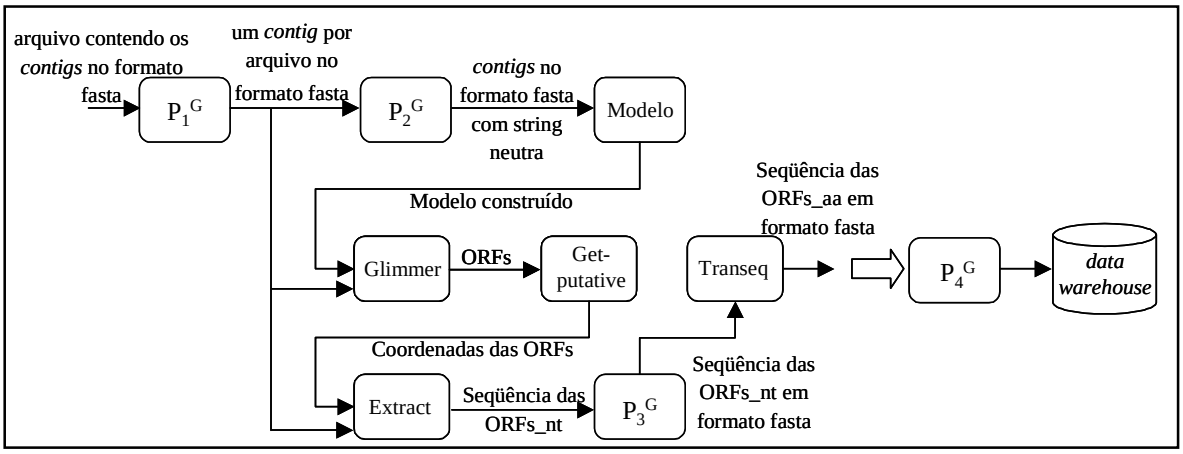

Figura 6 – *Workflow* seqüencial do Glimmer

Novamente, como no *workflow* do tRNAscan-SE, o *parser* P_1^G transforma o arquivo que contém todos os *contigs* no formato fasta para *n* arquivos no formato fasta com um único *contig* por arquivo. O próximo passo, executado pelo *parser* P_2^G, é unir todos os *contigs* em um único arquivo, colocando-se entre eles uma string neutra (por exemplo, "TGATTGATTGATCAATCAATCA") que tira a possibilidade de se encontrar ORFs na junção de dois *contigs*. Este procedimento é realizado em genomas que não estão totalmente seqüenciados. Constrói-se agora o Modelo de *Markov* Interpolado para os genes para que o programa Glimmer seja executado. Assim, para cada contig, executa-se o Glimmer com o objetivo de encontrar as ORFs, em seguida, o programa Get-putative com o objetivo de encontrar as coordenadas das ORFs e o programa Extract, que, a partir das coordenadas, extrai a seqüência das ORFs em nucleotídeos (ORFs_nt) encontradas nos respectivos *contigs*. As ORFs obtidas da saída do Extract não estão no formato fasta. Assim, o *parser* P_3^G transforma essas seqüências para o formato acima mencionado. Um outro passo importante é traduzir as ORFs cujas seqüências estão em nucleotídeos (ORFs_nt) para aminoácidos (ORFs_aa). Este passo é realizado pelo programa EMBOSS Transeq [RLB00] e corresponde também ao fluxo 6 na Figura 3. Este fluxo está embutido no *workflow* seqüencial do Glimmer porque durante a construção do XML e posterior inserção no *data warehouse* realizada pelo *parser* P_4^G, existe a necessidade de se obter as ORFs com as seqüências tanto em nucleotídeos quanto em aminoácidos.

O objetivo agora é procurar padrões importantes nas ORFs, como os terminadores e os sítios de ligações de ribossomos, representados na Figura 3, respectivamente, pelos fluxos 4 e 5. O *workflow* seqüencial do Transterm pode ser expandido como mostra a Figura 7.

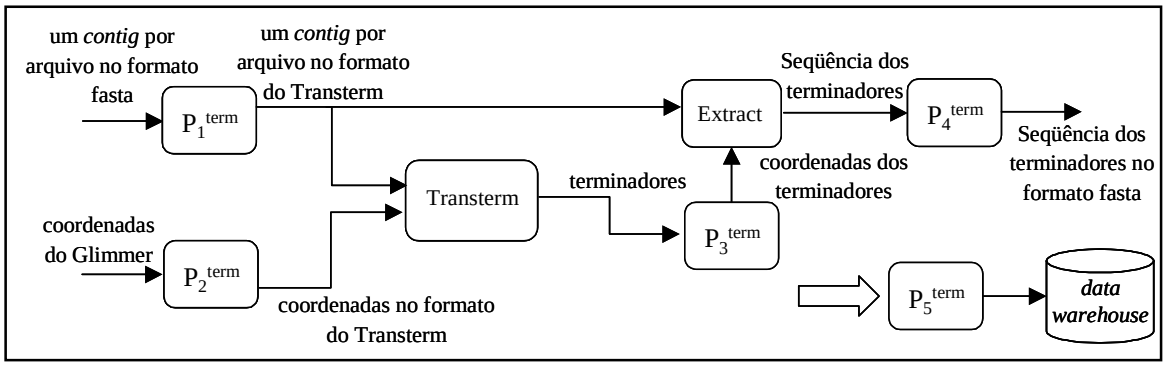

Figura 7 – *Workflow* seqüencial do Transterm

O programa Transterm necessita, para a sua execução, que os *contigs* e as coordenadas do Glimmer estejam em um formato específico. Assim, os *parsers* P_1^{term} e P_2^{term} transformam os *contigs* e as coordenadas do Glimmer, respectivamente, em um formato apropriado, que é "entendido" pelo Transterm. Tendo como entrada a saída do Transterm, isto é, os terminadores encontrados, o *parser* P_3^{term} obtém as coordenadas dos terminadores. Em seguida, com as coordenadas e os *contigs*, o Extract extrai a seqüência dos terminadores encontrados nas ORFs que não estão no formato fasta. O próximo passo, realizado pelo *parser* P_4^{term} é transformar a seqüência dos terminadores em formato fasta. Por último, o *parser* P_5^{term} insere os terminadores no *data warehouse* do BioNotes.

A expansão do *workflow* seqüencial do aplicativo que identifica os sítios de ligação dos ribossomos (RBSFinder) pode ser visualizada na Figura 8.

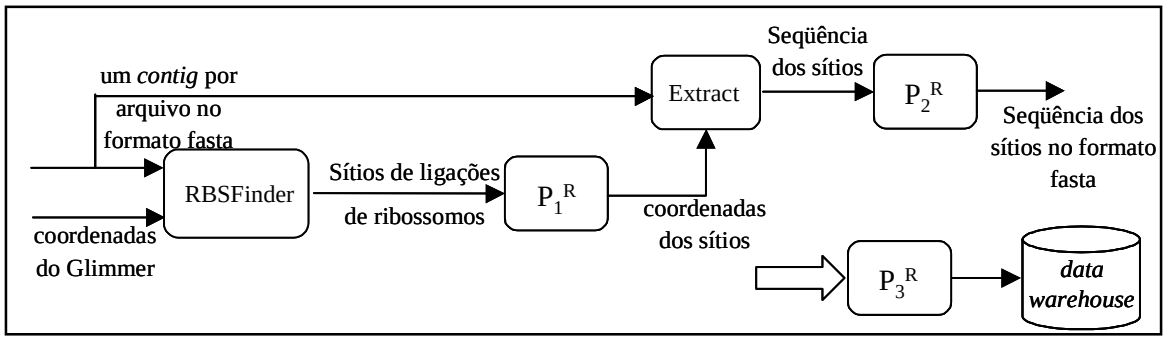

Figura 8 – *Workflow* seqüencial do RBSFinder

Primeiramente executa-se o RBSFinder tendo como entrada os *contigs* e as coordenadas do Glimmer. A partir da saída do RBSFinder, o *parser* P_1^R obtém as coordenadas dos sítios de ligações de ribossomos. Em seguida, executa-se o Extract tendo como entrada os *contigs* e as coordenadas dos sítios. As seqüências dos sítios são então transformadas em formato fasta pelo *parser* P_2^R e, por último, o parser P_3^R insere os sítios de ligações de ribossomos encontrados nas ORFs no *data warehouse* do BioNotes.

O fluxo 6, já explicado no *workflow* seqüencial do Glimmer, é composto somente pelo programa EMBOSS Transeq e é responsável por traduzir as ORFs de nucleotídeos para aminoácidos. As ORFs traduzidas são agora comparadas com milhares de seqüências de uma fonte ou base de dados com o objetivo de descobrir similaridades entre as seqüências. A base de dados contém, usualmente, várias seqüências no formato fasta (arquivo texto), não existindo nenhum tipo de SGBD envolvido [C02]. Ao descobrir seqüências similares, busca-se o conhecimento das funcionalidades das ORFs, por exemplo. Esta é uma das principais operações realizadas em biologia computacional e dependendo do ambiente de processamento, da quantidade de seqüências de entrada, da similaridade com as seqüências da base de dados e do tamanho da base utilizada, pode-se tornar uma operação com um alto custo computacional, demandando dias ou até semanas.

A busca de seqüências similares pode ser exemplificada pela expansão do *workflow* seqüencial do HMMPfam, que pode ser visualizada na Figura 9.

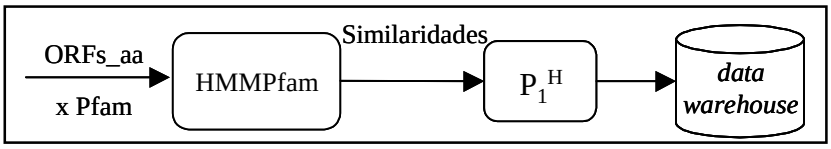

Figura 9 – *Workflow* seqüencial do HMMPfam

O HMMPfam faz parte de um *software* de Modelos de *Markov* Escondidos denominado HMMER [E98] e pesquisa por similaridades entre uma ou mais seqüências de entrada e uma base de dados HMM. Executa-se o HMMPfam tendo como entrada as ORFs cujas seqüências são formadas por aminoácidos e a base de dados de modelos de domínios de proteínas conhecidas denominada Pfam [BBC+02]. As similaridades encontradas com a base de dados são então transformadas para XML e inseridas no *data warehouse* do BioNotes pelo *parser* P_1^H.

Outra forma de buscar similaridades é feita através do *workflow* seqüencial do BLAST, que é muito parecido com o do HMMPfam e é mostrado na Figura 10.

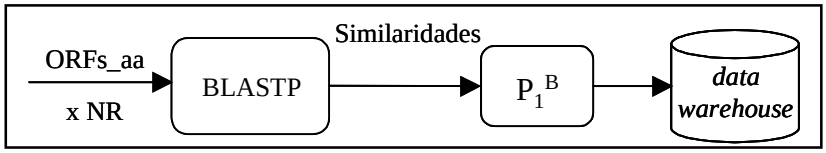

Figura 10 – *Workflow* seqüencial do BLAST

Executa-se o tipo de BLAST denominado BLASTP, cuja seqüência de entrada está na forma de aminoácidos (ORFs_aa) e cuja base de dados também está na forma de aminoácidos. A base de dados NR contém a tradução de toda a base de dados de nucleotídeos do GenBank (do NCBI - *National Center of Biotechnology Information)*, e outras bases, incluindo a SWISS-PROT [BBA+03], de forma não redundante [C02]. As similaridades encontradas são transformadas para XML e inseridas no *data warehouse* do BioNotes através do *parser* P_1^B.

Pode-se visualizar os aplicativos no BioNotes como uma caixa preta. Assim, a inserção de um novo aplicativo no *workflow* é uma operação relativamente simples, bastando-se, para isso, fazer um estudo sobre o programa procurando conhecer o formato de sua entrada e saída, parâmetros de execução e construir *parsers* de acordo com as necessidades do programa e para a inserção do padrão biológico em questão no *data warehouse*.

Em genomas ainda incompletos, uma nova execução do *workflow* configura uma nova versão do organismo no BioNotes. Com novos *reads* sendo seqüenciados, a execução do Phrap provavelmente formará novos *contigs*. Assim, programas como o tRNAscan-SE e o Glimmer que possuem como entrada os *contigs*, obterão resultados diferentes. É claro que o restante dos programas, que possuem como entrada as ORFs, também terão resultados diferentes. As anotações manuais geradas pela comunidade de usuários, que são dependentes das anotações automáticas poderão ser modificadas [LSC03][1].

Uma questão que surge é de quanto em quanto tempo o *workflow* deve ser executado. Isto depende da necessidade da comunidade de usuários. O BioNotes foi projetado para que o *workflow* seja executado quantas vezes e quando se queira [LSC03][1]. Contudo, o tempo gasto para que as anotações automáticas da *Glucona* estejam no *data warehouse* do BioNotes e sejam disponibilizadas para os usuários, é da ordem de algumas semanas,

principalmente devido aos programas como o HMMPfam e o BLAST que tentam buscar similaridades em um banco de seqüências, tornando-se, assim, um gargalo para todo o sistema. Este problema também impossibilita o estudo de organismos que necessitam de gerar anotações automáticas quase que diariamente. Portanto, algum tipo de técnica especial, como o paralelismo, deve ser aplicada para melhorar o desempenho da atualização das anotações automáticas no BioNotes.

3. – Workflow Paralelo da Glucona

3.1 – Alguns Conceitos em Computação Paralela

Atualmente, aplicações na área científica como problemas de otimização, simulações numéricas de sistemas complexos (por exemplo, na previsão do tempo), projetos Genoma, e aplicações na área comercial como sistema bancário e serviços para a *Internet* manipulam uma enorme quantidade de informações, exigindo, para a sua execução, máquinas cada vez mais rápidas e com maior capacidade de processamento. Para que haja um aumento no poder computacional, não basta obter processadores cada vez mais rápidos, pois limitações físicas no "ciclo de *clock*" do processador e os altos custos dos projetos *VLSI (Very Large Scale Integration)* impossibilitam o aumento da velocidade do processador além de certos limites. Assim, na tentativa de ultrapassar essas dificuldades, novas arquiteturas foram surgindo, introduzindo o conceito de paralelismo na computação [MRR02].

Segundo *Ian Foster* [F95], um computador paralelo é um conjunto de processadores que trabalham cooperativamente para resolver um problema computacional. Esta definição é bastante ampla e inclui multiprocessadores vetoriais (um pequeno número de processadores vetoriais de alta performance), processadores massivamente paralelos (centenas ou milhões de processadores conectados via memória compartilhada ou distribuída) e computadores em rede (*workstations* conectadas por uma rede de média/alta velocidade, funcionando como uma máquina virtual paralela de alta performance) [MRR02].

Do ponto de vista de organização da memória, uma máquina paralela pode ser dividida basicamente em memória compartilhada e memória distribuída [MRR02]. A Figura 11 mostra a arquitetura de memória compartilhada, onde todos os processadores (P_1,P_2,...,P_n) podem acessar, através de uma conexão de alta velocidade, todos os módulos de memória.

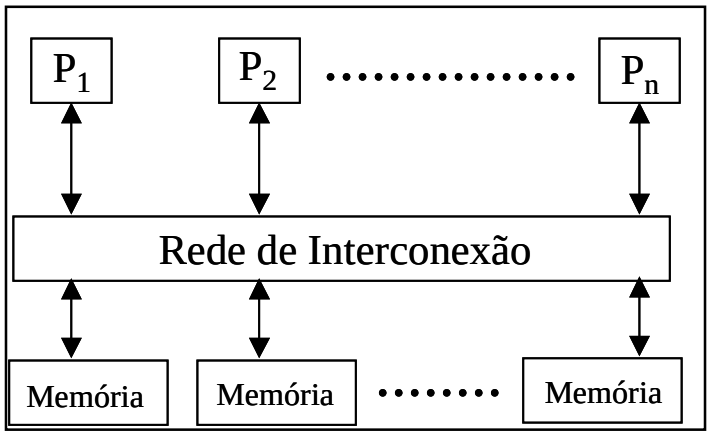

Figura 11 – Arquitetura de memória compartilhada [MRR02]

A comunicação entre os processadores ocorre através de operações de leitura/escrita na memória compartilhada. Assim, essa arquitetura possui como principal vantagem a rapidez de acesso aos dados. Porém, a escalabilidade do sistema é limitada a poucas dezenas de processadores devido à necessidade de ligar cada processador à uma dada unidade de memória. Também, o compartilhamento de memória pode levar a conflitos de acesso, diminuindo a performance do sistema. Algumas extensões podem ser realizadas nessa arquitetura com o objetivo de eliminar as desvantagens acima, como, por exemplo, adicionar memória local em cada processador [MRR02].

A Figura 12 mostra a arquitetura de memória distribuída, onde a memória é fisicamente distribuída entre os processadores e cada processador pode acessar somente a sua memória.

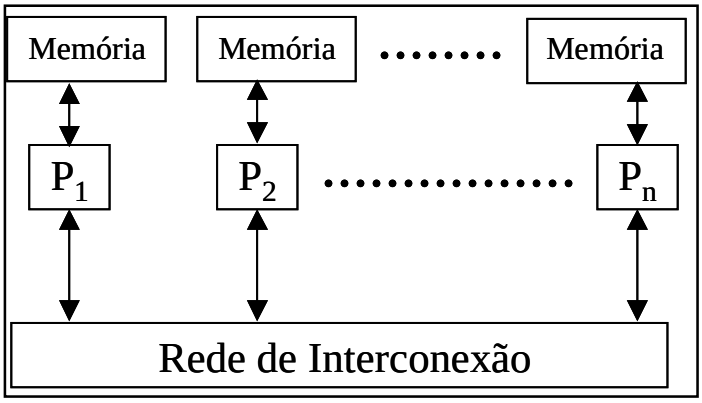

Figura 12 – Arquitetura de memória distribuída [MRR02]

A comunicação entre os processadores é realizada através de mensagens passadas através da rede de interconexão. Um exemplo dessa arquitetura é o *cluster* de computadores. Um *cluster* basicamente é uma coleção de PCs ou estações de trabalho conectados por uma rede, montados com componentes comerciais padrão [RR99].

Atualmente, tem-se tornado uma alternativa de baixo custo para processamento paralelo devido ao aumento de desempenho e confiabilidade dos PCs, estações de trabalho e redes de computadores [RR99], apresentando uma boa relação de custo/performance quando comparado com outras máquinas paralelas [MRR02]. Outras importantes vantagens desse sistema em se tratando de processamento paralelo são [RR99]: podem ser montados rapidamente, tornam fácil a adição de novos processadores (escalabilidade) e a facilidade de atualização de qualquer componente do sistema (processadores, rede ou *software*).

Para a troca de informações entre os processadores, existem dois modos de comunicação básicos: memória compartilhada e troca de mensagens [MRR02]. No modelo de comunicação de memória compartilhada, as tarefas compartilham um espaço de endereçamento comum, podendo acessar os dados de forma assíncrona. Vários mecanismos tais como *locks* e semáforos podem ser utilizados para controle do acesso à memória compartilhada. Este modelo pode ser diretamente utilizado em diferentes máquinas paralelas de memória compartilhada, mas pode também ser simulado em máquinas de memória distribuída [MRR02]. No modelo de troca de mensagens, processadores se comunicam através do envio e recebimento de mensagens em modo bloqueante (o processador interrompe o processamento de uma determinada atividade) ou não-bloqueante (o processador não interrompe o processamento de uma determinada atividade) [MRR02]. Este modelo pode ser diretamente utilizado em máquinas de memória distribuída, mas pode também ser simulado em máquinas de memória compartilhada [MRR02].

Dois tipos básicos de paralelismo podem ser explorados no desenvolvimento de programas paralelos: paralelismo de dados e paralelismo funcional [MRR02]. No primeiro, o mesmo conjunto de instruções é aplicado em múltiplos itens de uma estrutura de dados, como por exemplo, “adicione 2 a todos os elementos de um vetor” ou “aumente o salário de todos os empregados com 5 anos de serviço” [F95]. Este modelo pode ser facilmente implementado em máquinas de memória compartilhada. Em máquinas de memória distribuída, a localização dos dados é uma importante característica a ser considerada [MRR02]. No paralelismo funcional, cada tarefa pode executar um diferente conjunto de funções/código de forma assíncrona. Características importantes são a localização dos dados e a quantidade de processamento em cada tarefa [MRR02]. Essas técnicas são complementares e podem ser aplicadas em conjunto

em diferentes partes de um mesmo problema ou separadamente para a obtenção de soluções alternativas [F95].

Existe um número crescente de ferramentas que fornecem suporte para o desenvolvimento de programas paralelos. Essas ferramentas são essencialmente linguagens de programação seqüenciais ampliadas por um conjunto especial de chamadas ao sistema tais como primitivas de baixo-nível para troca de mensagens, criação e sincronização de processos, exclusão mútua e diversas outras funções [MRR02].

MPI (Message Passing Interface) surgiu da necessidade de definir uma interface de programação paralela comum, de modo a evitar a proliferação de programas escritos em diferentes "dialetos" [RR99]. Esse padrão foi introduzido pelo *MPI Fórum* em maio de 1994 e atualizado em junho de 1995 e envolveu aproximadamente 40 pessoas de 60 instituições, entre elas, vendedores de *hardware*, pesquisadores, acadêmicos, desenvolvedores de bibliotecas de *software* e usuários. *MPI* inclui a definição de interfaces de um conjunto de rotinas para comunicação por troca de mensagens, não definindo assim um ambiente de programação paralela completo. Por esta razão, *MPI* não manipula características tais como a estruturação e a depuração do programa paralelo, não fornece qualquer definição para suporte tolerante à falhas e assume que o ambiente computacional seja confiável [MRR02]. Desde a sua publicação, diversas implementações já foram desenvolvidas, sendo várias delas disponíveis publicamente tais como *mpich* [MP04] e *LAM* [LAM04]. Todas elas são baseadas em uma máquina virtual paralela, composta por vários computadores heterogêneos conectados (*workstations*, PCs, multiprocessadores) e cada um executa um processo para controlar a troca de mensagens entre as máquinas [MRR02]. O núcleo do *MPI* é formado por rotinas de comunicação entre pares de processos (*send* e *receive*). Além disso, o *MPI* define outras rotinas como a de comunicação e sincronização em grupo [RR95].

3.2 – Execução Paralela do BLAST em um Cluster de Computadores

Esta seção está baseada no trabalho realizado em [C02] que estudou a aplicação de técnicas de projetos de bancos de dados distribuídos na execução paralela do BLAST em uma máquina de arquitetura de memória distribuída.

Os algoritmos da família BLAST são os mais utilizados na biologia computacional para a operação de comparação de seqüências. Contudo, dependendo de diversos fatores como o tamanho da base de dados, a similaridade entre as seqüências, entre outros, pode se tornar uma operação bastante custosa, demandando horas ou até mesmo dias para a sua execução. Uma alternativa para melhorar a performance de execução do BLAST é a utilização de um ambiente paralelo. Assim, em [C02] são apresentados resultados práticos de implementações em um *Cluster* de PCs utilizando-se a versão WU-BLAST (BLAST da Universidade *Washington*) sem quaisquer modificações e a interface *MPI*.

Basicamente, dois esquemas de execução do BLAST em máquinas de memória distribuída foram apresentados [C02]: no primeiro, a base de dados está replicada em todos os nós e no segundo a base de dados é particionada e seus fragmentos alocados a diferentes nós. A Figura 13 mostra a estratégia com a replicação da base de dados em todos os nós, onde um esquema mestre-escravo é proposto. O nó mestre é responsável por distribuir as seqüências de entrada para os nós escravos (rotulados com os números 1,2,...,N), alocando, assim, tarefas aos mesmos. Uma tarefa realizada por um nó escravo compreende a execução de um processo BLAST sobre a base de dados replicada (rotulada por R1, R2,...,RN). Portanto, em cada nó escravo, o algoritmo BLAST foi devidamente instalado, configurado e deixado pronto para execução. Para que haja um equilíbrio de cargas, isto é, para que a carga de trabalho dos nós escravos seja semelhante, o nó mestre distribui as tarefas através de uma estratégia circular, isto é, a primeira seqüência é atribuída ao primeiro nó escravo, a segunda seqüência é atribuída ao segundo nó escravo, e assim sucessivamente. Caso o número de seqüências seja maior que o número de nós, ocorre uma nova atribuição ao primeiro nó escravo, ao segundo nó escravo, e assim por diante até que todas elas tenham sido distribuídas. Para um conjunto de k seqüências de entrada e N nós, o nó mestre enviará k mensagens. Após a distribuição, ocorre a execução de uma operação BLAST em cada nó escravo para cada seqüência, sendo os resultados enviados ao nó mestre que fica responsável pela montagem de um único resultado de saída [C02].

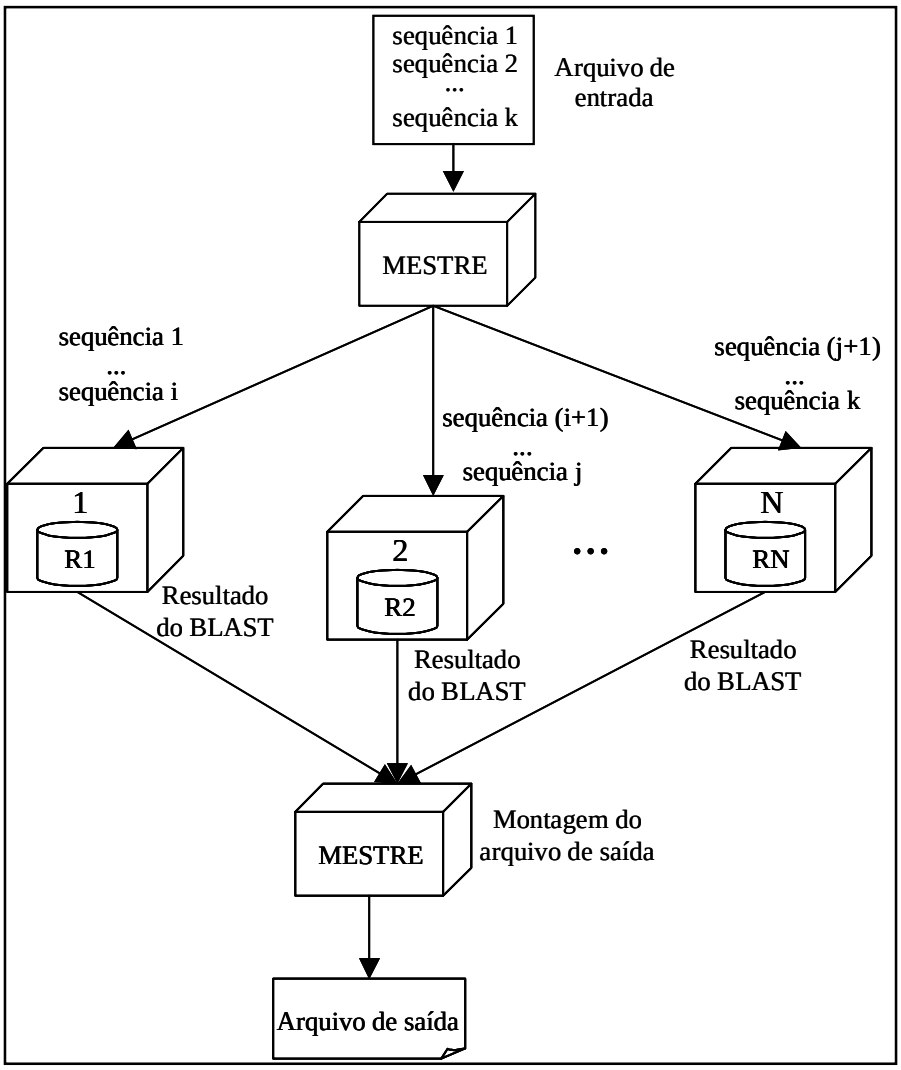

Figura 13 – Estratégia com replicação da base de dados [C02]

A Figura 14 mostra a estratégia com a fragmentação da base de dados. Aqui também um esquema mestre-escravo é proposto. Novamente, o nó mestre é responsável por distribuir as seqüências e os nós escravos são responsáveis pela execução da operação BLAST, agora sobre uma base de dados fragmentada (rotulada por F1, F2,...,FN), isto é, sobre uma parte da base de dados e não toda ela. O principal problema agora é como gerar os fragmentos. A estratégia adotada é gerar vários arquivos com um conjunto de identificadores/seqüência, obtendo assim subconjuntos disjuntos do arquivo original. Os arquivos gerados são os fragmentos alocados aos nós escravos. A distribuição de tarefas é diferente da estratégia replicada, pois pelo fato da base ser fragmentada, cada seqüência de entrada deve ser executada em cada nó escravo. Assim, para um conjunto de k seqüências de entrada e N nós, o nó mestre enviará kN mensagens. Após a execução, os resultados são enviados ao nó mestre que é responsável pela montagem do resultado final, sendo essa montagem mais complexa do que no caso da base replicada,

pois, para cada seqüência, existirão *N* resultados do BLAST, onde *N* corresponde ao número de fragmentos da base de dados [C02].

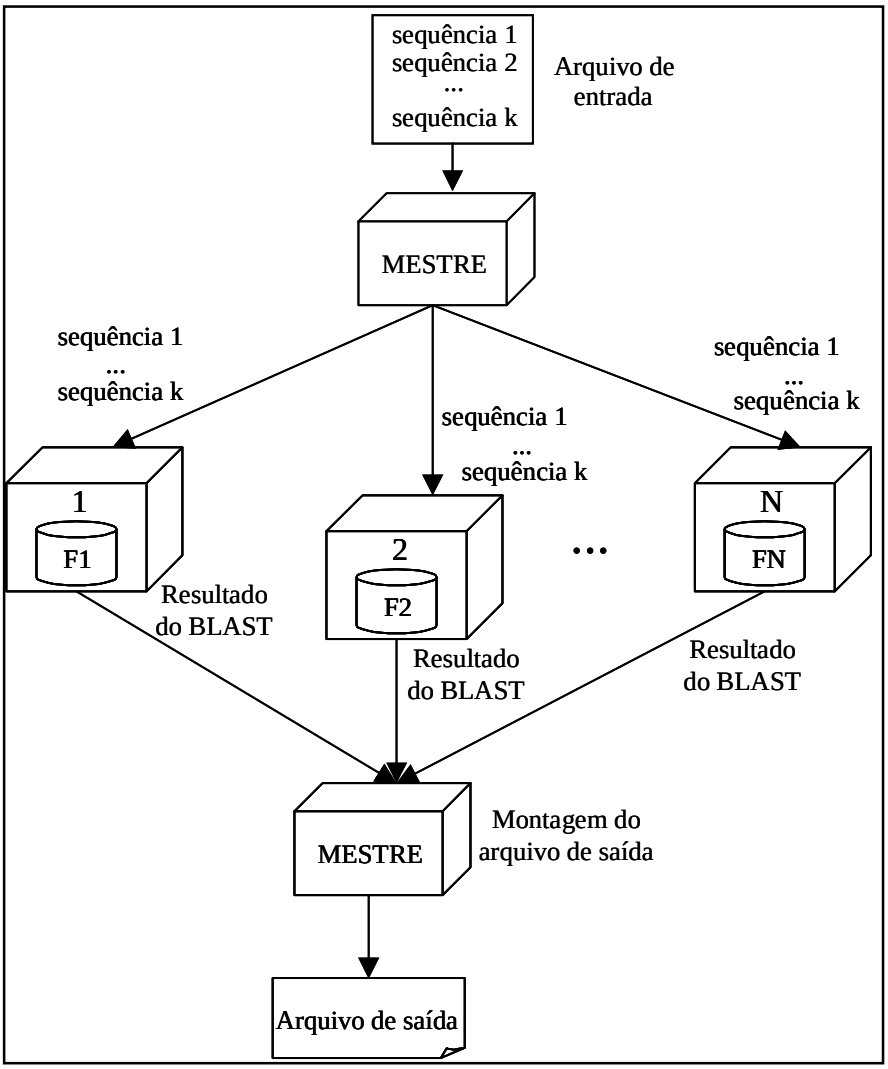

Figura 14 – Estratégia com fragmentação da base de dados [C02]

Foram realizados diversos testes que permitiram comparações de desempenho dos esquemas com replicação e fragmentação da base de dados, variando-se o número de nós escravos, o tamanho das seqüências de entrada, as estratégias de fragmentação, o tamanho da base de dados, entre outros. Observou-se, que, em geral, em grandes bases de dados, o desempenho da estratégia com fragmentação foi superior ao da estratégia com replicação e que, em pequenas bases de dados, o inverso ocorre porque existe um menor volume de comunicação na replicação do que na fragmentação e a montagem dos resultados é bem mais simples.

3.3 – Propostas de Paralelização do Workflow Seqüencial da Glucona

Esta seção tem como objetivo propor quatro alternativas para a paralelização do *workflow* seqüencial da *Glucona* em um ambiente de memória distribuída. A idéia é tentar diminuir o tempo de execução dos aplicativos e, conseqüentemente, diminuir o tempo necessário para que as anotações automáticas da *Glucona* estejam disponíveis para toda a comunidade. As próximas seções mostram as alternativas propostas e uma comparação entre as mesmas.

3.3.1 – Alternativa 1: Esquema Mestre Escravo com Base de Dados Replicada

A Figura 15 descreve a primeira alternativa de paralelização, onde um esquema mestre-escravo é proposto e as bases de dados do HMMPfam e do BLAST estão replicadas em todos os nós escravos (rotulados com os números 1,2, ..., N).

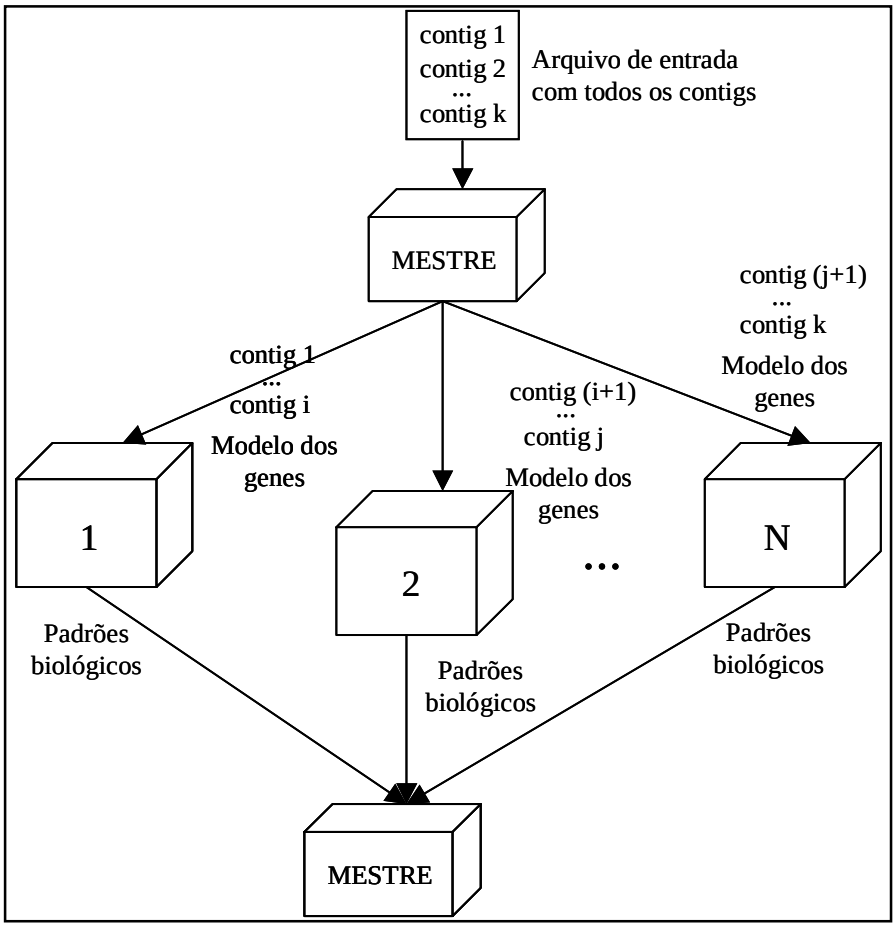

Figura 15 – *Workflow* paralelo da *Glucona* – Alternativa 1

O nó mestre recebe como entrada o arquivo com todos os *k contigs* de uma versão da *Glucona* e o transforma em *k* arquivos (um *contig* por arquivo) que serão distribuídos

através de uma estratégia circular aos *N* nós escravos, tentando manter a mesma carga de trabalho nos mesmos. O nó mestre também é responsável pela construção do modelo dos genes e pela distribuição aos *N* nós escravos para que o Glimmer seja executado. As tarefas executadas pelo nó mestre são detalhadas na Figura 16. Os *parsers* P_1^G, P_1^{tRNA}, P_2^G e Modelo são os mesmos descritos na seção 2.3.

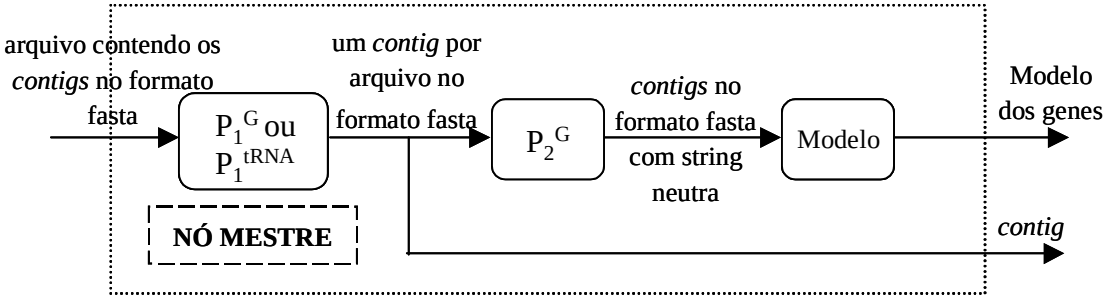

Figura 16 – Tarefas executadas pelo nó mestre

Cada nó escravo é responsável pela execução dos aplicativos do *workflow* da *Glucona*; assim, todos os programas, tais como tRNAscan-SE, Glimmer, Get-Putative, Transterm, BLAST, entre outros, deverão ser instalados, configurados e deixados prontos para execução em todos os nós escravos. A Figura 17 mostra o esquema que deve ser seguido para que um determinado *contig* seja executado em cada nó escravo.

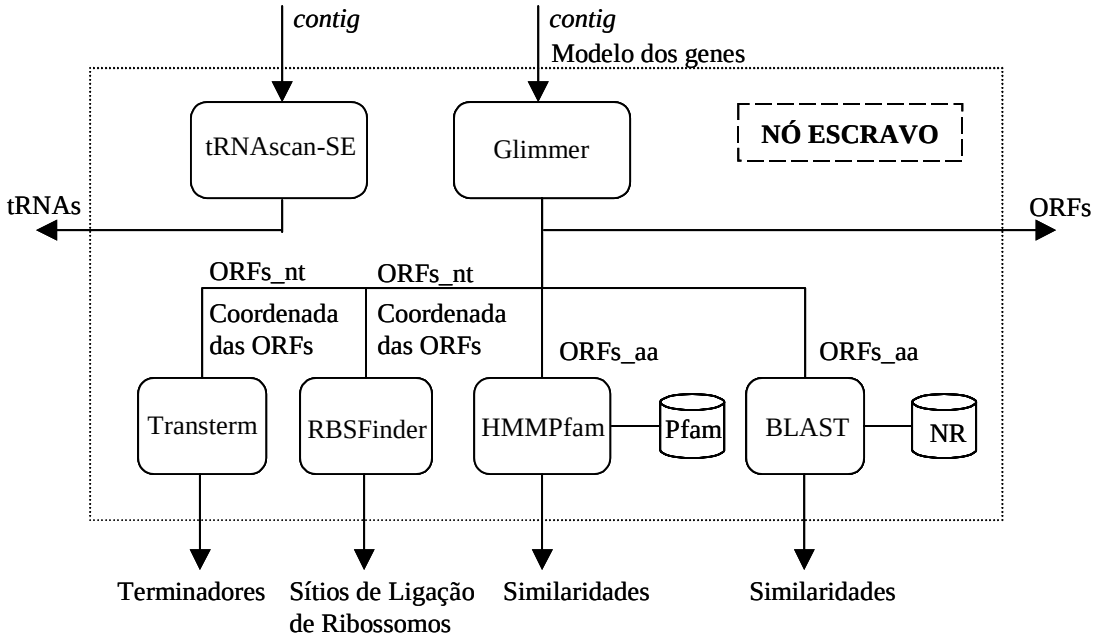

Figura 17 – Esquema para que um *contig* seja executado em cada nó escravo

Ao receber o *contig* e o modelo dos genes, o nó escravo pode executar o tRNAscan-SE e o Glimmer. Nesse momento, ocorre uma concorrência pelo processador no nó escravo. Deve-se salientar que alguns testes devem ser realizados com o objetivo de verificar qual a ordem de execução dos aplicativos trará um maior benefício, isto é, um

menor tempo de execução, e se o mesmo é significante ou não. Primeiramente será descrito o tRNAscan-SE e, logo, em seguida, o Glimmer. A Figura 18 mostra as tarefas que devem ser executadas por cada nó escravo com o objetivo de encontrar tRNAs no *contig* de entrada, correspondente ao fluxo (2b) do *workflow* seqüencial do tRNAscan-SE, mostrado na seção 2.3. Caso tRNAs sejam encontrados, eles serão enviados ao nó mestre.

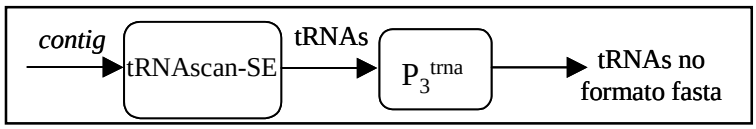

Figura 18 – Tarefas realizadas pelo nó escravo para executar o tRNAscan-SE

Já a Figura 19 mostra as tarefas que devem ser executadas por cada nó escravo com o objetivo de encontrar ORFs no *contig* de entrada, o que corresponde à parte do *workflow* seqüencial do Glimmer, pois a construção do modelo dos genes é realizada pelo nó mestre e distribuída para todos os nós escravos. As ORFs encontradas são enviadas para o nó mestre (ORFs_nt e ORFs_aa) e uma cópia das mesmas permanece no nó escravo para que os programas que possuam as ORFs como entrada possam ser executados. É claro que, se nenhuma ORF for encontrada no *contig*, é interrompida a execução do *workflow* e o nó escravo correspondente pode executar os aplicativos em um novo *contig*.

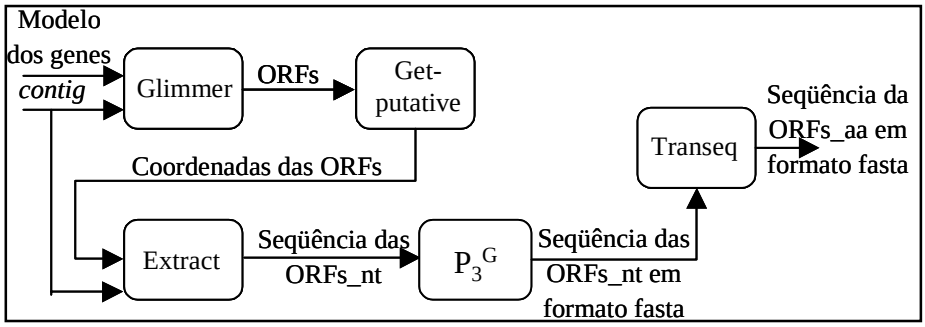

Figura 19 – Tarefas realizadas pelo nó escravo para executar o Glimmer

Caso contrário, pode-se continuar a execução do *workflow*. Novamente, tem-se uma concorrência no nó escravo, contudo, mais complexa que a anterior (entre o tRNAsca-SE e o Glimmer) porque existe um número maior de programas e um único *contig* pode conter várias ORFs. Assim, testes devem ser realizados com o intuito de verificar qual é a melhor ordem de execução dos programas. A continuação da descrição dos programas na Figura 17 será realizada da esquerda para à direita.

As tarefas realizadas para executar o Transterm e o RBSFinder são idênticas às tarefas do *workflow* seqüencial dos respectivos programas, mostrados na seção 2.3. A única diferença é que os *parsers* P_5^{term} e P_3^R, responsáveis pela transformação dos resultados do Transterm e do RBSFinder em *XML* e posterior inserção no *data warehouse*, não farão parte da execução do *workflow* paralelo em cada nó escravo (Figuras 20 e 21).

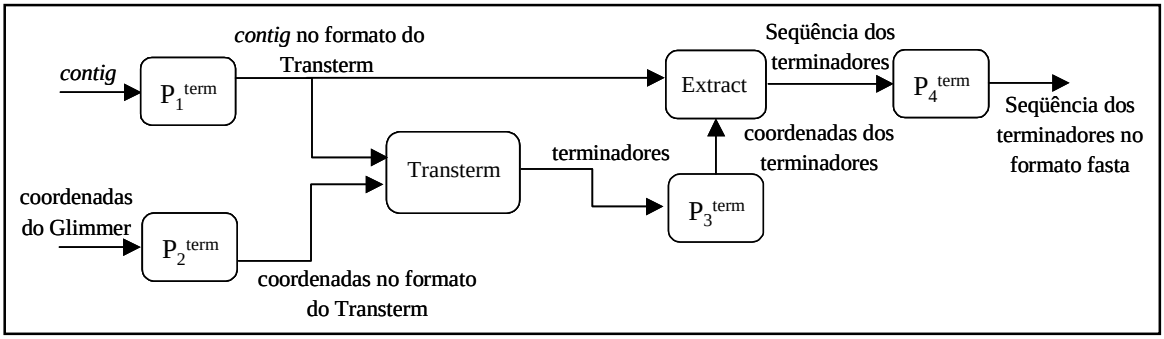

Figura 20 – Tarefas realizadas pelo nó escravo para executar o Transterm

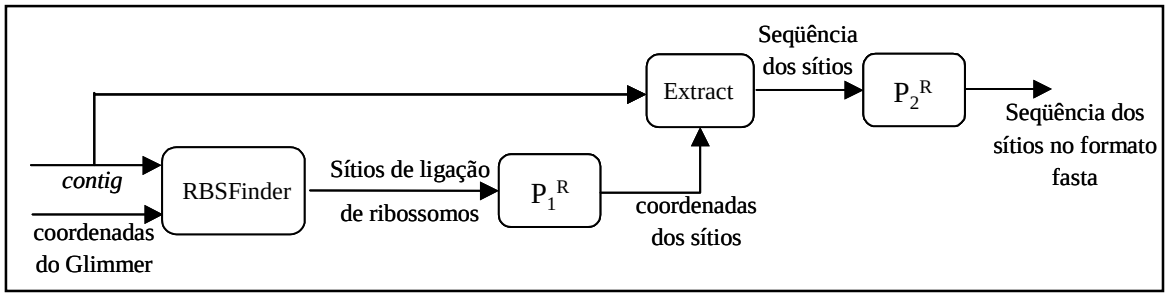

Figura 21 – Tarefas realizadas pelo nó escravo para executar o RBSFinder

A única tarefa realizada para a obtenção das similaridades com as bases de dados Pfam e NR é a execução do HMMPfam e do BLAST, respectivamente, como mostra as Figuras 22 e 23.

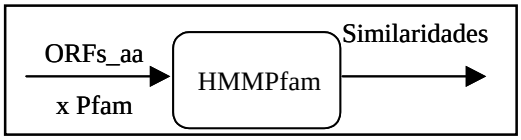

Figura 22 – Tarefas realizadas pelo nó escravo para executar o HMMPfam

Figura 23 – Tarefas realizadas pelo nó escravo para executar o BLAST

Essas duas operações demandam a maior quantidade de tempo na execução do *workflow* seqüencial da *Glucona* e, nessa alternativa, cada nó escravo possui uma cópia completa das bases de dados (estratégia replicada), o que torna a tarefa de programação um pouco mais simples.

3.3.2 – Alternativa 2: Esquema Mestre Escravo com Base de Dados Fragmentada

O BLAST obteve melhores resultados com grandes bases de dados utilizando a estratégia fragmentada [C02]. Assim, a segunda alternativa proposta (Figura 24) baseia-se na fragmentação das bases de dados Pfam e NR com respectiva alocação de parte das bases a cada nó escravo.

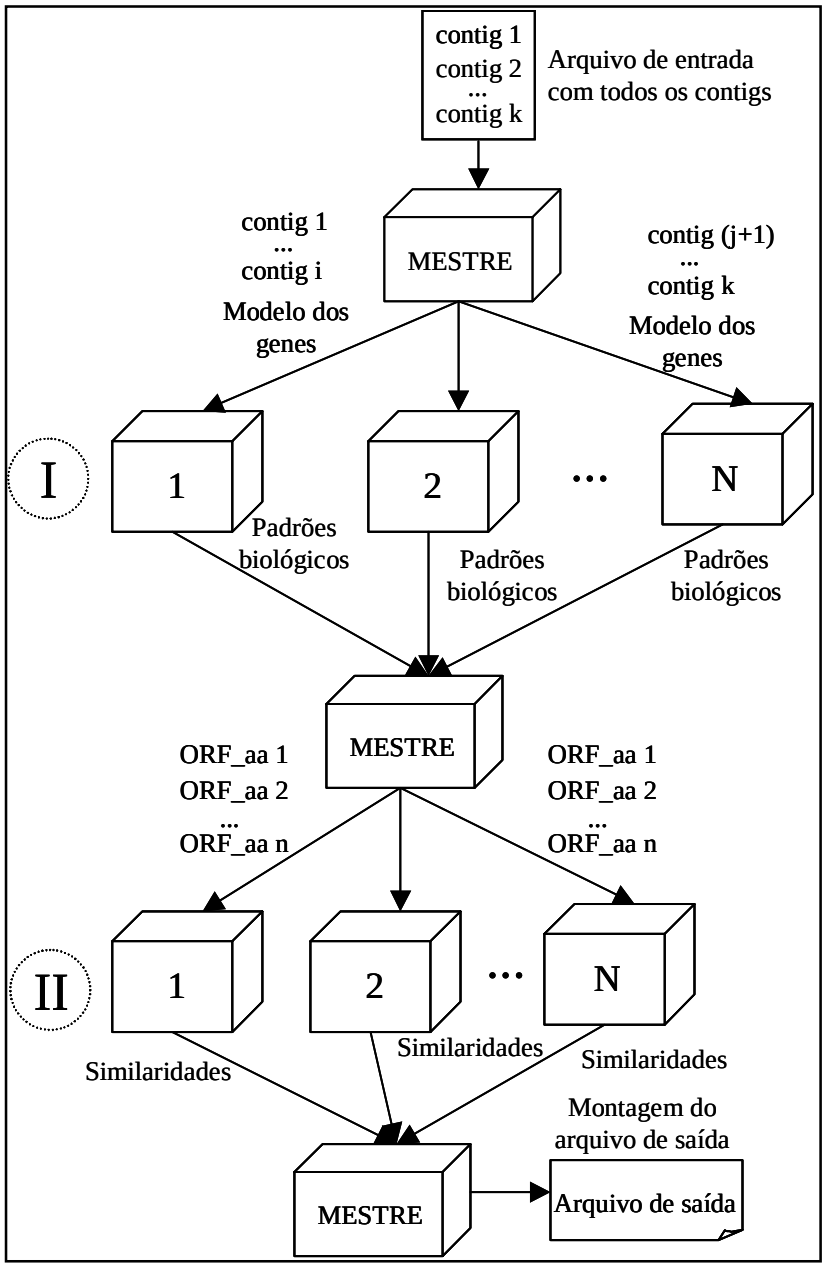

Figura 24 – *Workflow* paralelo da *Glucona* – Alternativa 2

Em um primeiro momento, como na alternativa 1, o nó mestre distribui todos os *contigs* e o modelo para o Glimmer para cada nó escravo (parte I na Figura 24). Novamente, todos os programas devem estar prontos para execução em cada nó, porém, os aplicativos responsáveis pela busca de similaridades serão executados posteriormente,

pois, como a base está fragmentada e por questões de corretude do resultado final, é obrigatório que todas as seqüências de entrada (ORFs_aa) sejam enviadas para cada nó escravo. Assim, o nó mestre ficará responsável por receber todos os padrões biológicos (tRNAs, ORFs_nt, ORFs_aa, terminadores e sítios de ligação de ribossomos) como resultado da execução dos programas e distribuir as *n* ORFs_aa encontradas pelo Glimmer para cada nó escravo (parte II na Figura 24). Existirão *N* resultados (similaridades) para cada seqüência de entrada, onde *N* indica a quantidade de fragmentos utilizados. Outra tarefa realizada pelo nó mestre é montar um arquivo de saída para cada seqüência de entrada a partir dos *N* resultados dos programas HMMPfam e BLAST (parte II na Figura 24).

Uma questão importante para essa alternativa é a utilização de um bom critério para a geração dos fragmentos. O critério utilizado em [C02] faz com que os fragmentos tenham um número total de seqüências o mais próximo possível. Dois métodos foram testados [C02]: Seqüências de Comprimentos Homogêneos, onde os fragmentos são gerados de forma que as seqüências de menor comprimento estejam posicionadas no primeiro fragmento, as seqüências seguintes estejam no segundo fragmento e assim por diante e Fragmentos de Totais Homogêneos que distribui as seqüências pelos fragmentos aplicando-se uma estratégia circular no conjunto de seqüências já ordenadas pelo tamanho. Esse segundo método obteve um melhor equilíbrio de carga entre os nós e um menor tempo de execução [C02]. Outro ponto importante a ser testado é o quanto o custo de comunicação afeta o desempenho do sistema já que um maior número de mensagens será enviado entre o nó mestre e os nós escravos e a concorrência entre os programas aplicativos.

3.3.3 – Alternativa 3: Esquema Mestre Escravo com Base de Dados Replicada e Divisão dos Programas

Observa-se, nas duas alternativas anteriores, um possível excesso de trabalho em cada nó escravo, já que a busca por similaridades é realizada contra grandes bases de dados. Uma outra proposta seria, então, separar os programas HMMPfam e BLAST dos outros aplicativos, isto é, alguns nós escravos seriam responsáveis pela execução do tRNAscan-SE, Glimmer, Transterm e RBSFinder e outros nós ficariam responsáveis exclusivamente pela busca de similaridades. Assim, como terceira alternativa, os

programas são divididos entre os diversos nós escravos e ocorre a replicação da base de dados.

Primeiramente, o nó mestre enviaria os *contigs* e o modelo para os nós escravos responsáveis pela execução do tRNAscan-SE, Glimmer, Transterm e RBSFinder e receberia os padrões biológicos como resultado. Em seguida, o nó mestre distribuiria, através de uma estratégia circular, as *n* ORFs_aa aos nós escravos responsáveis pela execução do HMMPfam e BLAST e receberia as similaridades com as respectivas bases de dados como resultado. Assim, a execução do *workflow* da alternativa 3 seria muito parecida com a execução do *workflow* da alternativa 2 mostrada na Figura 24. A primeira diferença estaria na divisão dos programas aplicativos entre os diversos nós escravos: na Parte I da Figura 24, os programas tRNAscan-SE, Glimmer, Transterm e RBSFinder estariam instalados nos nós escravos, já na Parte II, somente os programas HMMPfam e BLAST estariam instalados nos nós; as outras duas diferenças estão relacionadas à replicação da base de dados: as *n* ORFs_aa encontradas não precisariam ser enviadas para cada nó escravo responsável pela execução do BLAST e do HMMPfam e o nó mestre não teria necessidade de montar um arquivo de saída para cada seqüência de entrada.

Questões como a concorrência entre os programas, o custo de comunicação, a quantidade de processadores alocados as duas classes de programas (que buscam similaridades ou não) devem ser cuidadosamente exploradas. Em especial, deve-se salientar que o tempo de execução dos programas que não buscam similaridades é bem menor do que o tempo de execução do BLAST e do HMMPfam. Assim, o número de processadores alocados a primeira classe deve ser bem menor do que o número de processadores alocados aos programas que buscam similaridades (segunda classe). Caso a carga de trabalho fique ainda excessiva nos nós escravos responsáveis pela busca de similaridades, pode-se ainda separar a execução do HMMPfam e do BLAST, com alguns nós executando-se exclusivamente o BLAST e outros o HMMPfam.

3.3.4 – Alternativa 4: Esquema Mestre Escravo com Base de Dados Fragmentada e Divisão dos Programas

Como última alternativa proposta, pode-se dividir os aplicativos entre os diversos nós escravos e ao invés de replicar as bases de dados, pode-se fragmentá-las. A principal

diferença com a alternativa anterior é que como parte das bases de dados estará em cada nó escravo, todas as seqüências de entrada (ORFs_aa) devem ser enviadas a esses nós e não parte delas, como acontece com a estratégia replicada. Além disso, para tornar o resultado dos programas que buscam similaridades correto, o nó mestre ficará responsável por montar o arquivo de saída para cada uma das seqüências de entrada. Assim, baseando-se na Figura 24 para melhor entendimento, a única diferença estaria na divisão dos programas aplicativos entre os diversos nós escravos: na Parte I estariam instalados os programas tRNAscan-SE, Glimmer, Transterm e RBSFinder e na Parte II os programas HMMPfam e BLAST.

As mesmas questões levantadas para a alternativa 3 podem também servir como parâmetros de desempenho para essa alternativa proposta, como custo de comunicação, concorrência entre os programas, entre outros, além do critério utilizado para a geração dos fragmentos. Aqui também pode-se separar os programas HMMPfam e BLAST caso a carga de trabalho seja muito grande nos nós escravos.

3.3.5 – Comparação entre as Alternativas

As alternativas que possuem as bases de dados replicadas (alternativas 1 e 3) são mais fáceis de implementar do que as que possuem a base fragmentada (alternativas 2 e 4), pois não precisam se preocupar em montar um arquivo para cada seqüência a partir dos N resultados, onde N representa o número de fragmentos utilizados. Comparando-se as alternativas 1 e 3, a primeira é de mais fácil implementação porque é composta de um único passo.

O fato de utilizar as bases de dados replicadas ou fragmentadas também influencia na quantidade de mensagens trocadas entre o nó mestre e os nós escravos (custos de comunicação). As alternativas cujas bases estão replicadas trocam menos informações do que as que possuem as bases fragmentadas, pois no último caso, todas as seqüências de entrada devem ser enviadas para todos os nós escravos, o que não acontece quando a base está replicada. Comparando-se as alternativas 1 e 3, a primeira possui um menor custo de comunicação porque é composta de um único passo.

Quanto à carga de trabalho, o nó mestre trabalha mais na estratégia fragmentada do que na estratégia replicada, pois possui a tarefa extra de montagem do resultado final e os

nós escravos trabalham mais nas estratégias que não possuem a divisão dos programas (alternativas 1 e 2) do que nas alternativas que dividem o trabalho (alternativas 3 e 4).

Levando-se em consideração os resultados mostrados em [C02] e os tamanhos das bases de dados, as estratégias fragmentadas podem obter melhores resultados do que as estratégias replicadas. É claro que os custos de comunicação possuem um importante papel nesse resultado final.

Em todas as alternativas, a operação de transformação em XML e inserção dos padrões biológicos (*contigs*, *reads*, *ORFs*, terminadores, entre outros) no *data warehouse* do BioNotes será realizada separadamente da execução do *workflow* paralelo no *cluster* de computadores. Posteriormente, a estratégia que obtiver melhor desempenho poderia incluir a operação de inserção dos padrões biológicos no *data warehouse*, bastando, para isso, acrescentar os respectivos *parsers* no fluxo de execução dos programas.

4 – Considerações Finais e Trabalhos Futuros

Este trabalho apresentou quatro propostas de paralelização de um *workflow* seqüencial em um ambiente de memória distribuída. Primeiramente mostrou-se todo o *workflow* seqüencial da bactéria *Gluconacetobacter diazotrophicus*, responsável pela anotação automática desse organismo no BioNotes. Contudo, o tempo necessário para que essas anotações estejam disponíveis para a comunidade de usuários, isto é, o tempo necessário para que o *workflow* seqüencial seja executado e os padrões biológicos estejam no *data warehouse* do BioNotes, é da ordem de semanas, principalmente devido aos programas HMMPfam e BLAST, pois o número de seqüências de entrada é relativamente grande e as respectivas bases de dados são imensas. Assim, quatro propostas de paralelização do *workflow* foram apresentadas na tentativa de melhorar o desempenho dessa tarefa e está baseada no trabalho desenvolvido em [C02], cujos autores apresentaram resultados de projeto de bancos de dados distribuídos para a execução paralela do BLAST em um cluster de PCs e obtiveram ganhos expressivos. As alternativas para o *workflow* paralelo incluem a execução do BLAST e de vários outros programas; portanto muito do que foi abordado em [C02] pôde ser aplicado aqui.

Em geral, as quatro alternativas apresentadas estão baseadas em um esquema mestre-escravo e se diferem pela facilidade de implementação, pela quantidade de tarefas ou carga de trabalho realizada no nó mestre e nos nós escravos, pelas estratégias para a manipulação das bases de dados nos diversos nós escravos (replicação e fragmentação) e pelo número de mensagens trocadas entre os componentes do sistema (custos de comunicação). Dependendo da alternativa, importantes questões devem ser testadas e respondidas. Assim, o próximo passo desse trabalho é a implementação dessas propostas em um *cluster* de computadores utilizando a interface *MPI* com o objetivo de compará-las e de apontar os principais parâmetros que afetam o desempenho de cada proposta e aquela que obteve um melhor desempenho levando-se, em consideração, os resultados da execução seqüencial do *workflow* da *Glucona*.

Esse trabalho abordou propostas de *workflow* paralelo para a bactéria *Gluconacetobacter diazotrophicus*. Contudo, qualquer outro organismo poderia se beneficiar das técnicas descritas, principalmente aqueles cujo fluxo de execução dos programas possuam muitas operações BLAST e que precisem de atualização quase que diária do *data* warehouse, isto é, que os seus aplicativos sejam executados quase que

diariamente. Esse é o caso do organismo *Rhodnius prolixus*, onde o BioNotes também está sendo utilizado para auxiliar no processo de anotação. Assim, uma implementação do *workflow* paralelo do *Rhodnius* poderia estar baseada nesse trabalho e no trabalho desenvolvido em [C02].

5 – Referências Bibliográficas

[AGM+90] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J., ***A basic local alignment search tool***, Journal of Molecular Biology, v. 215, pp. 403-410, 1990.

[BBA+03] Boeckmann, B., Bairoch, A., Apweiler, R., Blatter, M.C., Estreicher, A., Gasteiger, E., Martin, M. J., Michoud, K., O'Donovan, C., Phan, I., Pilbout, S., and Schneider, M., ***The SWISS-PROT protein knowledgebase and its supplement TrEMBL***, Nucleic Acids Research, v. 31, pp. 365-370, 2003.

[BBC+02] Bateman, A., Birney, E., Cerruti, L., Durbin, R., Etwiller, L., Eddy, S. R., Griffiths-Jones, S., Howe, K. L., Marshall, M., and Sonnhammer, E. L., ***The Pfam protein families database***, Nucleic Acids Research, v. 30, pp. 276–280, 2002.

[BKL+03] Benson, D. A. , Karsch-Mizrachi, I., Lipman, D. J., Ostell, J., and Wheeler, D. L., ***GenBank***, Nucleic Acids Research, v. 31, pp. 23-27, 2003.

[C02] Costa, R.L.C., ***Alocação de Dados e Distribuição de Carga para Execução Paralela da Estratégia BLAST de Comparação de Seqüências***, Dissertação de Mestrado, Departamento de Informática, PUC-Rio, Março 2002.

[CL03] Costa, R.L.C. and Lifschitz, S., ***Database Allocation Strategies for Parallel BLAST Evaluation on Clusters***, Distributed and Parallel Databases, v. 13, pp. 99-127, 2003.

[D01] Dowell, R., ***A Distributed Annotation System***, Technical Report, In partial fulfillment of the requirements for the degree of Master of Science (project option). Department of Computer Science, St. Louis, MO, February 2001.

[DB04] Department of Biochemistry and Molecular Biology, Oswaldo Cruz Institute, [on-line]: http://www.dbbm.fiocruz.br, April 2004.

[DHK+99] Delcher, A.L., Harmon, D., Kasif, S., White, O., and Salzberg, S.L., ***Improved microbial gene identification with GLIMMER***, Nucleic Acids Research, v. 27, n. 23, pp. 4636-4641, 1999.

[E98] Eddy, S. R., ***Profile hidden Markov models***, Bioinformatics, v. 14, pp. 755–763, 1998.

[EG98] Ewing, B. and Green, P., ***Base-Calling of Automated Sequencer Traces using Phred. II. Error Probabilities***, Genome Research, v. 8, pp.186-194, 1998.

[EHW+98] Ewing, B., Hillier, L., Wendl, M.C., and Green, P., ***Base-Calling of Automated Sequencer Traces using Phred. I. Accuracy Assessment***, Genome Research, v. 8, pp.175-185, 1998.

[F95] Foster, I., ***Designing and building parallel programs: Concepts and tools for parallel software engineering***, Addison-Wesley, 1995.

[FAH+01] Frishman, D., Albermann, K., Hani, J., Heumann, k., Metanomski, A., Zollner, A. and Mewes, H-W., ***Functional and structural genomics using PEDANT***, BioInformatics, v. 17, n. 1, pp. 44-57, 2001.

[FP03] Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro, Notícias da [on-line]: http://www.faperj.br/interna.phtml?obj_id=692, 2003.

[G03] Green, P., ***Documentation for Phrap***, [on-line]: http://bozeman.mbt.washington.edu/phraps.docs/phrap.html, March, 2003.

[H97] Harris, N. L., ***Genotator: a workbench for sequence annotation***, Genome Research, v. 7, n. 7, pp. 754-762, 1997.

[LAM04] LAM MPI [on-line]: http://www.mpi.nd.edu/lam, 2004.

[LE97] Lowe, T.M., and Eddy, S.R., ***tRNAscan-SE: a program for improved detection of transfer RNA genes in genomic sequence***, Nucleic Acids Research, v. 25, pp. 955-964, 1997.

[LSC03][1] Lemos, M., Seibel, L.F.B. e Casanova, M. A., ***Sistemas de Anotações em Biossequências***, Monografia de Ciência da Computação (PUC-RioInf.MCC04/03), Departamento de Informática, PUC-Rio, Março 2003.

[LSC03][2] Lemos, M., Seibel, L.F.B. and Casanova, M. A., ***BioNotes: A System for Biosequence Annotation***, 1st International Workshop on Biological Data Management, In conjunction with DEXA, Prague, Czech Republic, 2003.

[MLF+99] Möller, S., Leser, U., Fleischmann, W., and Apweiler, R., ***EDITtoTrEMBL: a distributed approach to high-quality automated protein sequence annotation***, BioInformatics, v. 15, n. 3, pp. 219-227, 1999.

[MP04] mpich [on-line]: http://www-unix.mcs.anl.gov/mpi/mpich, 2004.

[MRR02] Martins, S.L., Ribeiro, C.C. e Rodriguez, N.R., ***Parallel Computing Environments***, Handbook of Applied Optimization (P. Pardalos e M.G.C. Resende), pp. 1029-1043, Oxford, 2002

[R04] RIOGENE: Virtual Institute of Genomic Research, [on-line]: http://www.riogene.lncc.br/, April 2004.

[RLB00] Rice, P., Longden, I., and Bleasby, A., ***EMBOSS: the European Molecular Biology Open Software Suite***, Trends Genetics, v.16, n. 6, pp. 276-7, Juny 2000.

[RR99] Ribeiro, C.C. e Rodriguez, N.R., ***Otimização e Processamento Paralelo de Alto Desempenho***, Revista PUC Ciência, pp. 45-48, dezembro 1999.

[S01] Stein, L., ***Genome Annotation: From Sequence to Biology***, Nature Reviews - Genetics, v. 2, pp. 493-505, July 2001.

[SG03] Smit, A.F.A. and Green, P., ***RepeatMasker***, [on-line]: http://ftp.genome.washington.edu/RM/RepeatMasker.html, 2003.

[SL00] Seibel, L.F.B., ***Bio-AXS: Uma Arquitetura para Integração de Fontes de Dados e Aplicações de Biologia Molecular***, Tese de Doutorado, Departamento de Informática, PUC-Rio, Maio 2000.

[SL01] Seibel, L.F.B. and Lifschitz, S., ***A Genome Database Framework***, DEXA, pp. 319-329, 2001.

[SNN+02] Sakata, K., Nagamura, Y., Numa, H., Antonio, B.A., Nagasaki, H., Idonuma, A., Watanabe, W., Shimizu, Y., Horiuchi, I., Matsumoto, T., Sasaki, T. and Higo, K.:, ***RiceGAAS: an automated annotation system and database for rice genome sequence***, Nucleic Acids Research, v. 30 pp. 98-102, January 2002.

[SSC+94] Scharf, M., Scheneider, R., Casari, G., Bork, P., Valencia, A., Ouzounis, C. and Sander, C., ***GeneQuiz: a workbench for sequence analysis***, Intell. Syst. Molecular Biology, v. 2, pp. 348-353, 1994.

[SWW+96] Smith, R. F., Wiese, B. A., Wojzynski, M. K., Davison, D. B. and Worley, K. C., ***BCM search launcher - An integrated interface to molecular biology data base search and analysis services available on the World Wide Web***, Genome Research, v. 6, pp. 454-462, 1996.

[THG94] Thompson, J.D., Higgins, D.G., Gibson, T.J., ***CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice,*** Nucleic Acids Research. v. 22: pp. 4673-4680, 1994.

[TS01] Tusnády, G.E. and Simon I., ***The HMMTOP transmembrane topology prediction server***, Bioinformatics, v. 17, pp. 849-850, 2001.

[TS98] Tusnády, G.E. and Simon I., ***Principles Governing Amino Acid Composition of Integral Membrane Proteins: Applications to Topology Prediction***, Journal of Molecular Biology, v. 283, pp. 489-506, 1998.

[UF03] Universidade Federal do Rio de Janeiro, Notícias da [on-line]: http://www.bioinfo.ufrj.br/Imprensa/cienciahoje.html, 2003.

[WHA+02] Wu, C. H., Huang, H., Arminski, L., Castro-Alvear, J., Chen, Y., Hu, Z., Ledley, R. S., Lewis, K. C., Mewes, H.W., Orcutt, B. C., Suzek, B. E., Tsugita, A., Vinayaka, C. R., Yeh, L.S.L., Zhang, J., and Barker, W.C., ***The Protein Information Resource: an integrated public resource of functional annotation of proteins***, Nucleic Acids Research, v. 30, pp. 35-37, 2002.