Avaliação de desempenho do programa BLAST: análise de compactação de dados, gerência de memória e escalonamento de processos

Aluno: Diogo Gitahy

Orientador: Sérgio Lifschitz

Introdução

O BLAST é uma poderosa ferramenta usada na bioinformática para efetuar comparações entre biossequências. Sua execução pode demorar muito se levarmos em conta que existem bancos de dados com alguns gigabytes de tamanho.

Frequentemente na comparação de biossequências, parte do banco de dados é carregada na memória mas não é lida, desperdiçando tempo e capacidade de armazenamento que poderia estar sendo usada de maneira mais otimizada.

Ew comum também na utilização do BLAST que usuários distintos mandem sequências diferentes para serem comparadas com um mesmo banco de dados. No entanto, quando isso ocorre, um usuário precisa esperar que o outro termine o seu processo para começar a sua comparação caso a leitura da base de dados já esteja em andamento.

Pensando nisso, foram desenvolvidas algumas técnicas que visam a melhoria na eficiência de execucação da ferramenta BLAST. Para resolver o problema do carregamento excessivo em memória pensou-se em compactar as bases de dados e descompactá-las somente se necessário, apenas nos trechos a serem utilizados.

Em relação à execuções simultâneas, foi desenvolvido um driver capaz de controlar a disponibilização de dados para os processos, fazendo com que chamadas distintas pudessem começar a processar a partir de uma leitura já iniciada, comparando com a parte inicial posteriormente, otimizando o tempo.

Estratégias

A. Compactação de dados

Na tentativa de diminuir o número de operações de entrada e saída do BLAST, foi criada uma estratégia baseada em compactar as sequências a serem usadas para comparação,

pois assim aumenta-se a quantidade de informação disponível em memória ao mesmo tempo. Dessa forma, uma única leitura varre mais informação.

Em [1], foi utilizada uma forma de compactação baseada no algoritmo BWT, pois esse possui uma taxa de compressão comparável aos melhores e por apresentar um bom desempenho.

O processo segue a seguinte ordem: compactar o banco já formatado pelo FORMATDB antes da execução do BLAST e descomprimir em tempo de execução.

Infelizmente os resultados nos ambientes utilizados não foram satisfatórios. Nas tabelas abaixo podemos ver que o número de operações de E/S realmente diminuiu, mas o tempo de execução do programa não foi melhorado.

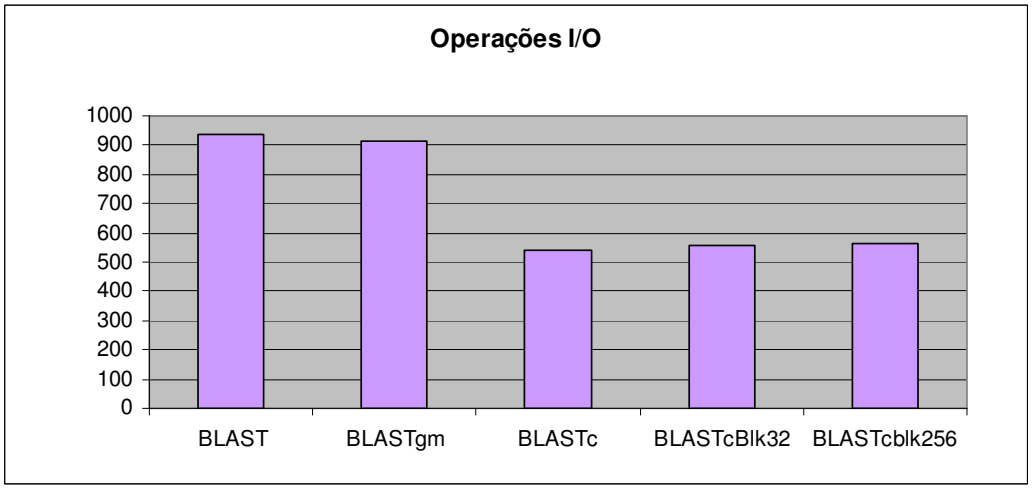

Figura 1 – Número de operações de E/S [1]

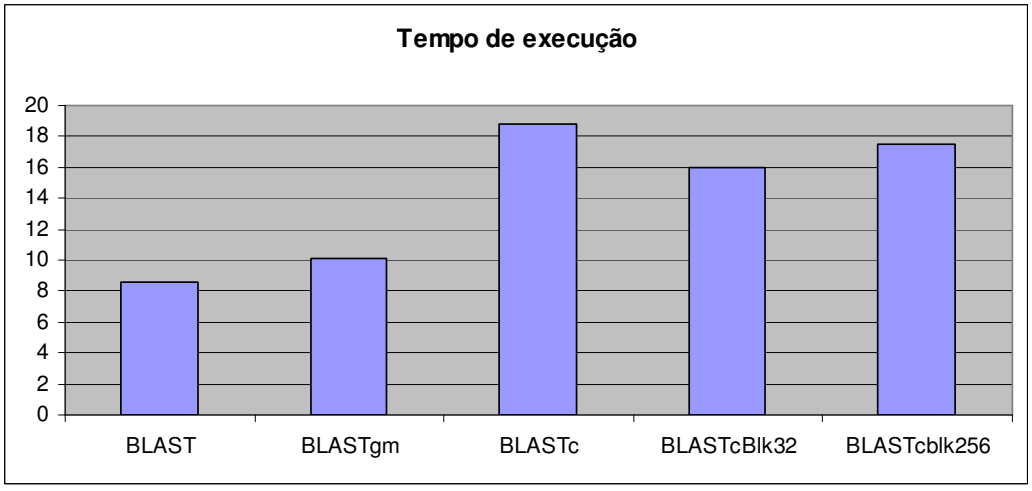

Figura 2 - Tempo necessário para o processamento [1]

Legenda: BLAST: BLAST normal;

BLASTgm: BLAST com gerência de memória;

BLASTc: BLAST com compactação;

BLASTcBlkX: BLAST com compactação utilizando blocos de X KB na compressão.

B. Controle de memória

BioProvider[2] é uma ferramenta utilizada como provedora de dados biológicos para programas como o BLAST. Seu objetivo é conseguido atravées de gerenciamento de buffer de maneira mais eficaz do que a realizada normalmente pelo sistema operacional. Além disso, ela realiza um escalonamento dos processos do mesmo.

A ferramenta foi implementada para Linux como módulo do kernel. Ela faz com que os arquivos de bancos de dados sejam substituídos por arquivos associados ao driver que, quando são chamados pelo BLAST, se associam a funções do driver. Essas funções realizam a comunicação entre o banco e o processo de comparação do programa. O esquema abaixo ilustra essa interação.

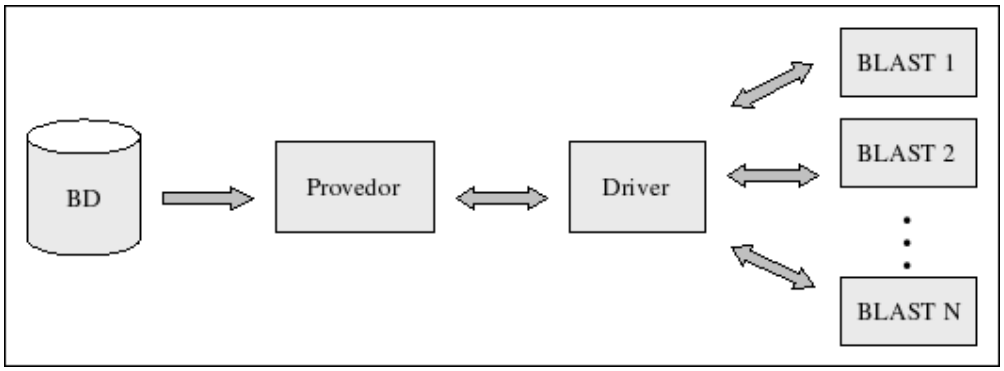

Figura 3 - Relação entre processos BLAST e o banco de dados, quando acionado o BioProvider [2]

Para evitar problemas com os ponteiros utilizados pelos arquivos de índice dos bancos de dados, se faz necessária a divisão deste em n partes. Esse número é determinado de acordo com o tamanho do buffer(ou anel) que vai ser utilisado. Após isso deve-se formatar essas partes de forma que fiquem ordenadas para que a leitura possa ser feita a partir de um ponto diferente do início. Por exemplo: vamos dividir um banco em três blocos(1,2 e 3) e aplicar o formatdb nos blocos concatenados para obtermos os arquivos de índice na sequência necessária(123, 231 e 312).

Apesar do alto custo de memória inicial, a estratégia é válida pois ela pode ser utilizada independente da versão do BLAST e independente da formatação dos arquivos de índice, já que só interessa o formato do arquivo de sequências.

Os resultados obtidos com essa ferramenta foram bastante satisfatórios como podemos perceber nas tabelas expostas abaixo.

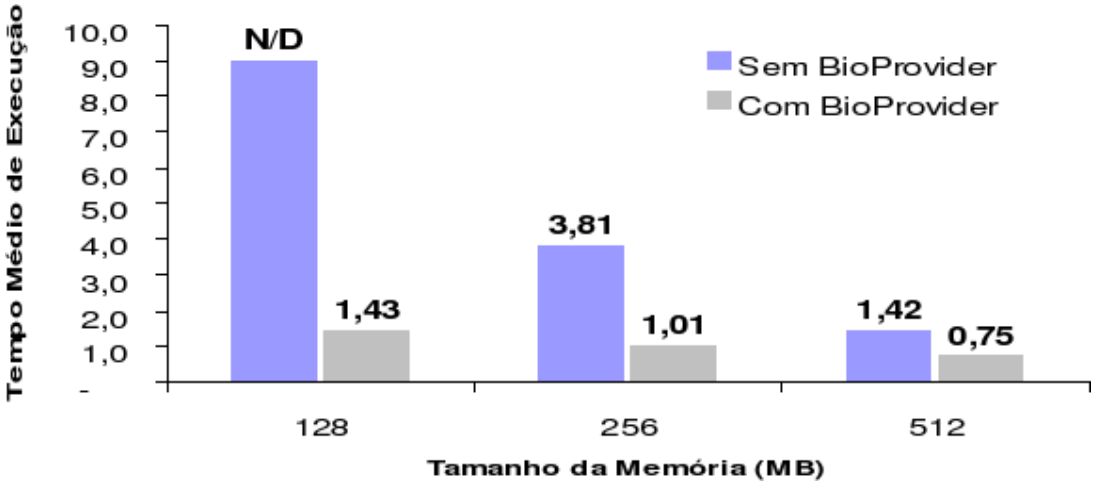

Figura 4 - Desempenho utilizando um banco de 1GB para diferentes tamanhos de memória disponível [2]

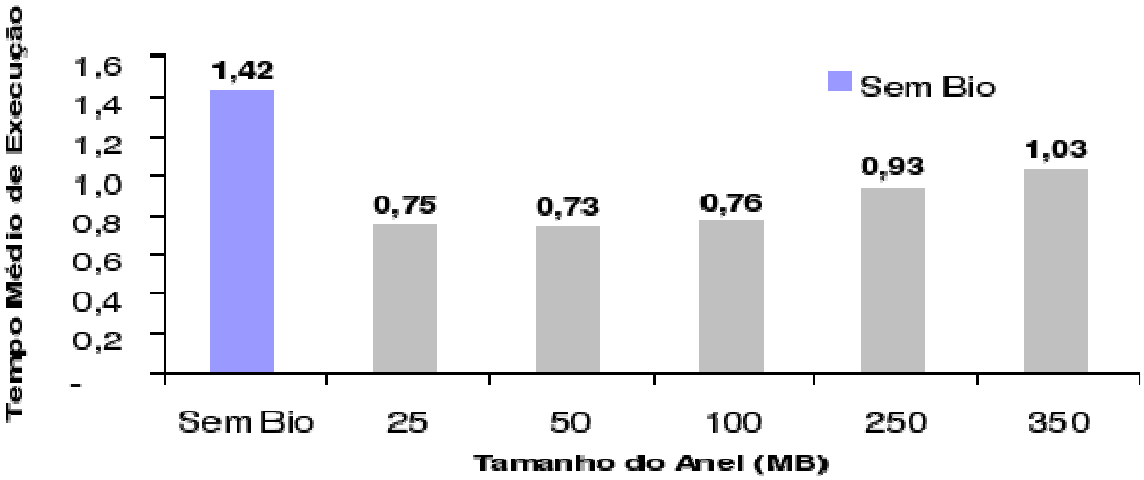

Figura 5 - Desempenho para um banco de 1GB variando o tamanho do anel utilizado, para memória disponível de 512MB [2]

C. Conclusão

A idéia apresentada em [1] merece ser estudada em outros ambientes e com outras estratégias de compactação, para podermos afirmar precisamente sobre a sua eficiência.

Já a estratégia criada em [2] já mostra-se eficaz mas precisa ser estudada mais a fundo para garantir que essa eficiência se mantém em diferentes ambientes. Além disso, pode-se perceber que o processo de preparação do ambiente para a utilização da ferramenta é bem custoso, podendo ser tratado de forma mais adequada.

Metodologia e Resultados

A. Compactação de dados

Em relação ao estudo apresentado em [1], foi proposto em [3] uma forma diferente de compactação e configurações diferentes de ambiente. Foi tirada a chamada à função BWT, já que essa é verdadeiramente útil apenas na compactação, e o mais importante a ser otimizado é a descompactação.

Para a realização de testes de forma mais eficientes, estes foram automatizados através de arquivos batches(*.bat). Esses arquivos têm como objetivo preparar o ambiente e iniciar o processo BLAST.

A preparação do ambiente consiste somente em compactar o banco de dados a ser usado. No entanto, para executar os processos é importante levar em consideração que se rodarmos testes com um mesmo banco de maneira consecutiva, estaremos viciando os resultados. Isso ocorre porque o banco já estaria armazenado na memória, nao precisando ser carregado do disco.

A forma de evitar tal problema foi criar um batch que “limpasse” a memória cache da máquina. Isso foi feito rodando o BLAST normal, ou seja, que não utiliza a compactação, com bancos que tivessem um tamanho suficientemente grande para cobrir a memória disponível.

Feito isso, rodaram-se batches que compactavam bancos de dados e chamavam o BLAST com compactação para diferentes parâmetros, como o tamanho de bloco usado, a máquina que o rodou, além de chamar o batch “de limpeza” entre os testes consecutivos.

Os resultados demonstrados em [3] não demonstram exatamente um padrão de melhoria dos processos com a compactação. Em alguns casos ocorreu melhora, mas em outros o BLAST normal continuou sendo mais eficiente. De qualquer forma as novas alterações foram bem sucedidas, já que em todos os casos o novo método de compactação foi mais eficiente que o anterior, como verifica-se abaixo.

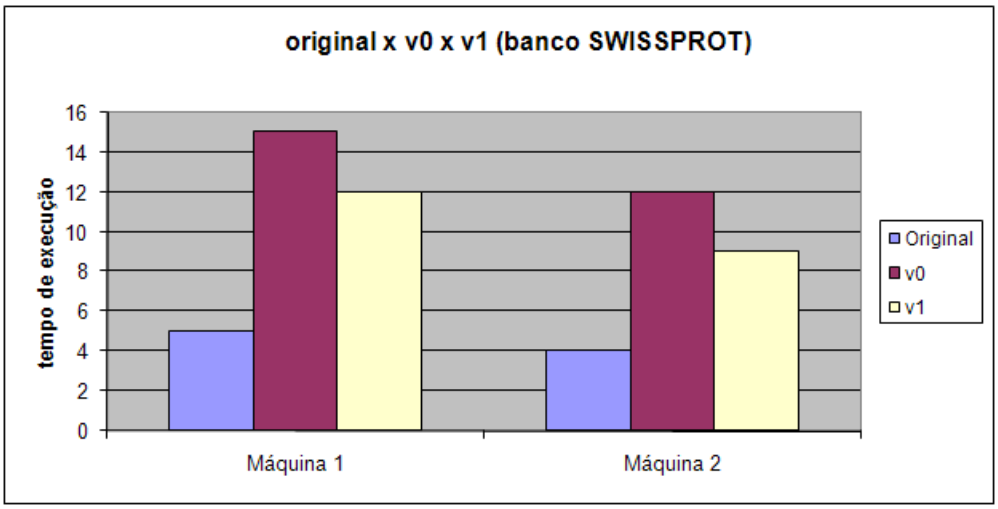

Figura 6 - Teste usando o blast original, o blast com BWT(v0) e o blast com compactação sem BWT(v1) [1]

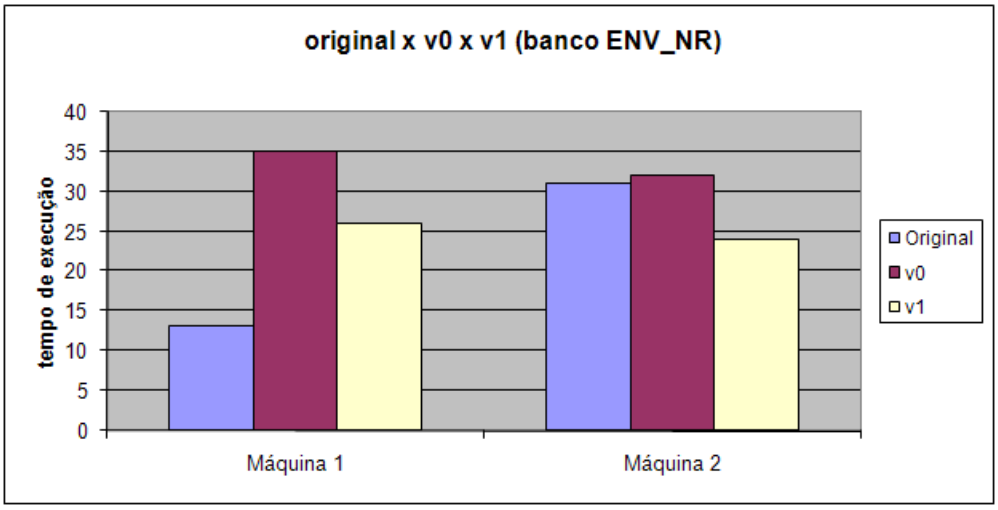

Figura 7 - Teste usando o blast original, o blast com BWT(v0) e o blast com compactação sem BWT(v1) [1]

B. Controle de memória

Quanto ao BioProvider[2], com a intenção de acabar com a "burocracia" necessária para configurar o ambiente de sua execução, aproveitando que o programa foi feito para Linux, foram criados scripts em BASH para automatizar todo o processo, facilitando a realização de testes.

Foram feitos três scripts de configuração de ambiente: um para formatar o banco de dados, outro para instalar o driver e ativar o BioProvider e um último para desativar o BioProvider e desinstalar o driver.

O primeiro foi separado dos demais por ser muito custoso e pela falta de necessidade de rodá-lo mais de uma vez para um mesmo conjunto de parâmetros. Esse script consiste em:

1) Buscar os parâmetros necessários para seu funcionamento. Essa busca é feita em um arquivo texto que o usuário deve preencher de acordo com seus dados. Um exemplo desse arquivo segue abaixo:

```
Diretorio do BioProvider = /home/BioProvider
Diretorio do blast = /home/ncbi/blast/build
Caminho completo do Banco de Dados Fasta = /home/bd/nr/nrfasta
Tamanho do anel em KB = 100000
```

2) Em seguida o script utiliza os dados fornecidos para aplicar o formatdb no banco fasta original, para poder descobrir o tamanho do arquivo .psq que será utilizado. Com esse tamanho e com o número de sequencias do banco, encontrado pelo get_num_seq do BioProvider, o script encontra e armazena as variáveis que dependem desses valores. Essas variáveis são: tamanho do anel, tamanho do .psq, número de sequências, número de blocos e número de sequências por bloco.

3) Após obter os valores necessários, o script cria o arquivo "config" recompilando o BioProvider para que ele leve em consideração os novos dados.

4) O passo seguinte, o mais custoso, consiste em criar os *n* blocos fasta com o copy_block do BioProvider, e concatená-los da maneira adequada, ou seja, formando fastas do tamanho original mas que comecem por partes diferentes do bloco e sigam a ordem certa(i.e. para 4 blocos serão criados 4 blocos fasta -1,2,3,4- que serão concatenados formando três fastas - 2341, 3412, 4123-, pois o fasta 1234 é o original). Ao final do processo cada fasta deve ser formatado pelo formatdb do BLAST. Os .psq's gerados podem ser excluídos, basta manter o original.

5) Por fim são removidos os blocos fasta já que eles só serviam para formarem os fastas maiores.

O segundo e o terceiro foram separados entre si para deixar a execução do BLAST em si a cargo do usuário, da maneira que melhor o convier. Além disso, ambos exigem privilégios de root, enquanto a execução não. O segundo consiste apenas em instalar o driver e obter seu número de processo para criar os special files requeridos pelo BioProvider, além de iniciar o próprio. O terceiro faz o inverso: finaliza o provedor, desinstala o driver e deleta os special files.

Os testes foram feitos através de um script que rodava processos concorrentes com diferentes sequências de entrada, simulando vários usuários diferentes. Os resultados estão sintetizados abaixo.

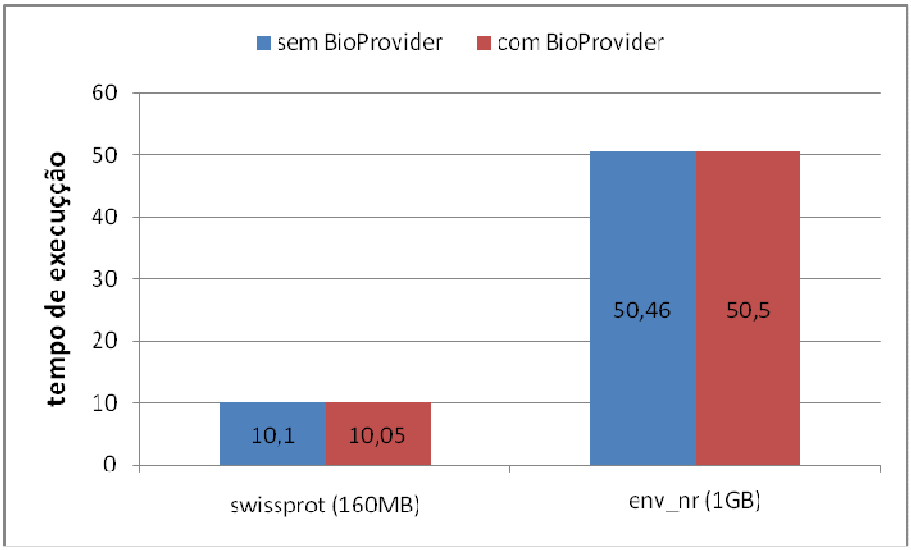

Figura 8 - testes do BioProvider com memória disponível de 1GB

Conclusões

De maneira geral, ficou claro que a automatização dos testes, mesmo que de maneira simplória com scripts ou batches, auxilia muito a realização de experimentos, tornando desnecessária a intervenção do usuário durante o trabalho da máquina.

Em relação a estratégia de compactação de dados, não foram obtidos resultados que satisfizessem totalmente a necessidade daqueles que atuam na área da bioinformática, mas ao menos houve um progresso, mostrando que com mais intervenções e mais testes, talvez seja possível melhorar o desempenho na comparação de biossequências.

Já no que diz respeito ao BioProvider, apesar de não parecerem bons, os resultados servem para comprovar um fato plausível se tomarmos o conceito do BioProvider: a ferramenta não tem vantagem em cima do BLAST comum se o banco de dados couber na memória do computador. Isso ocorre pois a vantagem do provedor é justamente otimizar a parte do disco que vai ser disponibilizada na memória para ser usada. Se tudo que for necessário do disco já estiver na memória, o próprio sistema operacional otimiza de maneira satisfatória.

Bibliografia

1- ROSA, J. O. M. **Um estudo de compactação de dados para biossequências**. 2006. 135 f. Dissertação (Mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, Rio de Janeiro, 2006.

2- NORONHA, M.F. **Controle da execução e disponibilização de dados para aplicativos sobre sequências biológicas: o caso BLAST**. 2006. 83 f. Dissertação (Mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, Rio de Janeiro, 2006

3- Gomes, Luciana da Silva Almendra. Lifschitz, Sérgio. **Estudo de Compactação de Biosseqüências para Uso do Programa BLAST.** Rio de Janeiro, 2008. 31f. Relatório Final de Projeto Final – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.