3 Análise do programa BlastP

Este capítulo inicia-se com a apresentação das principais características do programa BLAST e continua com uma visão detalhada da implementação do NCBI-BLAST.

Também será mostrada a análise realizada sobre o desempenho do programa BlastP com relação às operações de acesso a dados em memória secundária.

3.1 Descrição do Funcionamento do BLAST

Existem duas implementações principais do BLAST, o NCBI-BLAST [Ncb05] e o WU-BLAST [WUB05]. O NCBI-BLAST é uma implementação mantida pelo Centro Nacional de Informações de Biotecnologia, nos EUA (*National Center for Biotechnology Information*, NCBI). O NCBI mantém bancos de dados públicos, conduz pesquisa em biologia computacional, desenvolve ferramentas de software para a análise de dados genômicos e dissemina informações biomédicas.

O WU-BLAST é a implementação mantida pela Universidade de Washington, nos EUA. A universidade mantém em domínio público apenas o código da versão 1.4 de seu programa e alguns executáveis mais recentes com funcionalidade reduzida. Atualmente o WU-BLAST está na versão 2.0, este de uso livre, mas de código não aberto.

De acordo ao tipo de comparação que o usuário deseja fazer, o programa BLAST se divide em cinco subprogramas [BKY03], como mostrado na Tabela 1:

BlastP	Utiliza como seqüência de entrada uma seqüência de aminoácidos contra um banco de dados de proteínas. Esse tipo de BLAST é muito utilizado quando se tem uma proteína e deseja-se saber se existem, em outros organismos, proteínas similares.
BlastN	Compara uma seqüência de nucleotídeos contra um banco de dados de nucleotídeos. O Blastn é muito utilizado para procurar seqüências de nucleotídeos que são muito conservados.
BlastX	Compara uma seqüência de nucleotídeos contra um banco de dados de proteínas. O Blastx ajuda na função de descobrir em qual quadro de leitura uma determinada seqüência de nucleotídeos deve ser traduzida em proteína, e em obter um valor biológico a essa possível proteína.
tBlastN	Utiliza uma seqüência de aminoácidos contra um banco de nucleotídeos. Esta ferramenta possibilita encontrar genes novos que ainda não foram descritos antes, isso ocorre devido ao grande numero de proteínas desconhecidas, principalmente proteínas de pequeno tamanho.
tBlastX	Utiliza como seqüência de entrada uma seqüência de nucleotídeos contra um banco de nucleotídeos. O tBlastx é a ferramenta ideal quando se tem uma seqüência de nucleotídeo desconhecida e quer se dar um valor biológico a essa seqüência.

Tabela 1 – Tipos de Programas BLAST

Os programas BLAST utilizam como entrada uma seqüência que será comparada com uma base de dados. A seqüência usualmente está representada como um arquivo no formato FASTA. Também poderá ser introduzida diretamente a seqüência de caracteres ou ainda um identificador NCBI válido, no caso da versão desenvolvida por este centro. A base de dados, contra a qual será comparada a seqüência de entrada, é arquivo texto no formato FASTA, sendo aceitos também arquivos no formato ASN.1 [ASN05]. A partir deste banco

de dados são gerados pelo menos três arquivos com um utilitário do BLAST chamado **FORMATDB**, em formato binário [NCB02] que serão realmente manipulados pelo programa.

A comparação de seqüências é realizada em três passos [BKY03], como mostrados na Tabela 2:

Passo 1
Construção da lista de palavras: são procuradas palavras de um tamanho **w** fixo que possuem pontuação no mínimo igual a um limite **T** quando alinhadas, sem buracos, com alguma palavra (também de tamanho **w**) da seqüência de consulta.
Passo 2
Determinados todos os casamentos exatos (*hits*) das palavras encontradas no passo anterior com as seqüências do banco de dados.
Passo 3
Extensão dos *hits*: cada casamento exato encontrado no passo anterior é estendido até que a sua pontuação caia um valor **X** abaixo da melhor pontuação encontrada para extensões menores.

Tabela 2 – Passos do Algoritmo BLAST

O banco de dados de seqüências é utilizado no segundo e terceiro passos do algoritmo do BLAST. No passo 1, a seqüência de entrada (de consulta) é lida para separar as porções ou palavras de tamanho **w**. No passo 2, o banco de seqüências é lido seqüencialmente e vai sendo colocado em uma estrutura de dados auxiliar em memória a posição inicial de cada hit encontrado. Esta posição é um número que indica o *offset* da palavra encontrada como *hit* dentro do arquivo. No terceiro passo, o banco de seqüências deve ser acessado, a partir da posição de cada *hit* e consumido à medida que as extensões são testadas.

O esforço computacional de execução do BLAST é normalmente menor do que o custo associado ao volume e característica dos dados. Para bancos de seqüências maiores, um número maior de operações de leitura/escritura ao disco é realizado, e pior o desempenho obtido. Além do tamanho, outra variável que influi diretamente no tempo de execução é o grau de similaridade, definido como a presença da mesma informação biológica entre seqüências de DNA.

Nesta seção será estudado com detalhe o funcionamento da versão do código do NCBI-BLAST disponibilizado em dezembro de 2005 [Ncb05], sendo testados um dos programas principais, BlastP, visando à análise aprofundada sobre as operações de E/S realizadas por este em cada passo do algoritmo. Essa análise permitirá detectar com precisão as características das operações de leitura e escrita realizadas pelo BLAST, tais como: freqüência de execução, ordem de execução, quantidade de dados recuperados por leitura, se realiza *prefetch* ou não, etc.

Foi escolhido o BlastP pois, dentre as distintas representações adotadas na formatação feita pelo FORMATDB, é usado uma taxa de compressão de 4:1 para bases de nucleotídeos, e 1:1 para bases de proteínas, o que torna o arquivo de proteínas mais provável de obter-se uma melhora com o uso de compactação.

3.2 Descrição da implementação do NCBI-BLAST

Nesta seção é introduzida a organização da família de programas **NCBI-BLAST** do ponto de vista da implementação. Esta organização é mostrada a partir dos módulos definidos na implementação do programa. Diagramas serão usados para ilustrar a relação entre os módulos e as principais funções implementadas em cada um.

Estes dados foram coletados mediante o estudo e acompanhamento passo a passo do código distribuído pelo NCBI, usando como ambiente de desenvolvimento o Microsoft Visual C++. Na distribuição do NCBI é possível ter acesso ao código fonte de várias ferramentas na mesma área de trabalho. São disponibilizados todos os subprogramas do BLAST além de outras versões como MEGABLAST [Ncb05], aplicativos como o FORMATDB e bibliotecas para tratamento de dados no formato ASN.1.

Para a execução passo a passo foi escolhido o subprograma BlastP, sendo testado inicialmente com os valores padrão dos parâmetros de entrada, o banco de seqüências de proteínas ecoli.aa, que contem traduções do genoma da *Escherichia-coli*[Ncb05b], e a seqüência de entrada, mostrados na Tabela 3 abaixo.

Parâmetros de entrada	
Programa	BlastP com parâmetros padrão.
Seqüência de entrada	MSYQEAMELSAEARAILPPENADLKTLVGNGYVVITLGLEIGADDYITKP FNPRELTIRARNLGSMPSQPDLTIIESSSQIAYKLSGRVASLHVPAVVSS ARLAMELQAEPIIRSNFYRGVIHPDTDPILLTLIDTLAGDGFGKLAPS
Banco de seqüências	Ecoli.aa – 4.289 seqüências; 1.358.990 total de bases

Tabela 3 - Configuração da execução do BlastP para estudo do código.

Na Figura 2, os módulos principais do BLAST são apresentados. Os módulos descritos aqui são: **ncbi**, **ncbitool**, **demo_blastall** e **ncbimain,** pois estes módulos contêm a parte do código em que são analisadas operações de E/S.

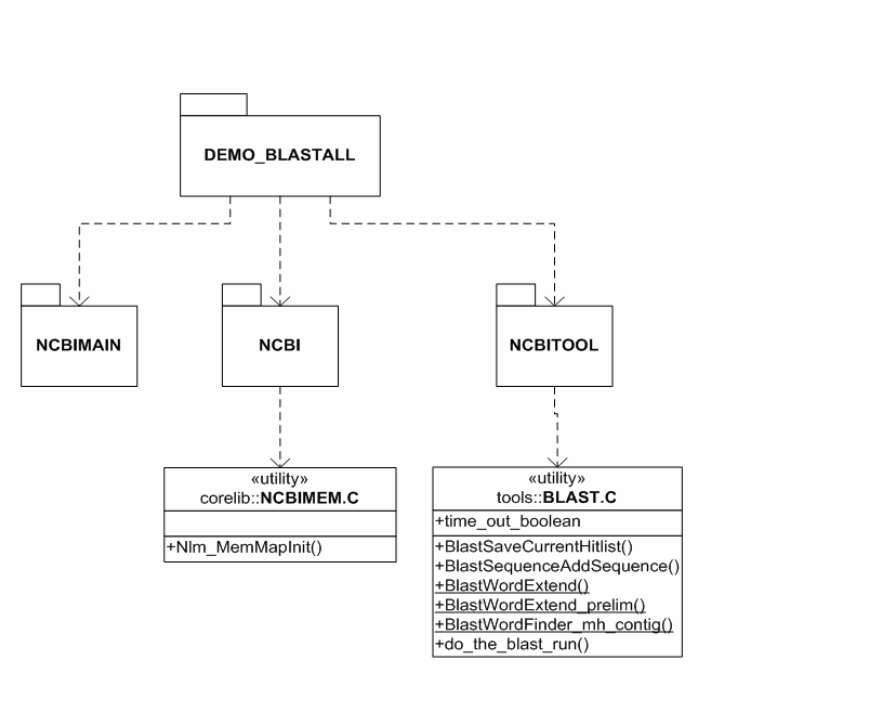

Figura 2 - Módulos Principais do NCBI-BLAST

O módulo ou projeto **ncbimain** contém, além da função principal de entrada ao programa, funções para receber os parâmetros de execução do BLAST.

A partir da função *main* do módulo **ncbimain**, é chamada a função *NLM_Main* do módulo **demo_blastall**. Esta função confere a validade dos parâmetros de execução, como a existência dos arquivos do banco de seqüências e arquivo com seqüência de entrada, e inicia a carga de estruturas

de dados utilizada durante toda a execução do BLAST com as opções de execução.

Este módulo inclui, entre outros, o arquivo com as funções que executam os três passos do algoritmo descrito na Figura 2. Para isto, utiliza procedimentos para gerencia de memória e arquivos disponibilizados pelos módulos **ncbi** e **ncbitool** respectivamente.

Entre outros, o módulo **ncbitool** contem o arquivo *readdb.c* no qual estão definidas as operações para manipulação de arquivos como abertura, leitura e fechamento. Estas funções foram especialmente desenvolvidas para tratar com arquivos seguindo um recurso de gerência de arquivos conhecido como arquivos mapeados em memória, *memory mapped files*, utilizado pelo BLAST. Este recurso foi incluído nas versões mais atuais do sistema como uma otimização no acesso aos arquivos, já que a gerência da alocação e transferências de dados passa a ser controlada pelo sistema operacional diretamente, e para a aplicação os acessos são similares a acessos a memória principal.

Outro arquivo bastante avaliado foi o arquivo *ncbimem.c*. Este arquivo é parte do módulo ncbitool. Os procedimentos para gerência de memória estão reunidos neste arquivo.

Implementação do algoritmo

As funções que implementam especificamente os três passos ao algoritmo fazem parte do arquivo *blast.c*. A comparação propriamente dita começa na função *BlastWordFinder_mh_contig*. Cada seqüência do banco de dados a ser analisada, chamada de *subject sequence* no momento de avaliação, é percorrida de palavra em palavra e testada contra a lista de palavras armazenada na tabela de busca (*lookup table*). Esta busca é inicializada com a chamada à função *lookup_find_init*, que procura a palavra inicial da seqüência na tabela. A partir daí, operações de deslocamento de bits e máscaras são usadas para ir descobrindo cada letra da seqüência e formando as palavras do tamanho definido pelo parâmetro de entrada W. Tais operações são necessárias já que, para cada letra, somente os cinco bits mais relevantes são usados para representá-las verdadeiramente. Assim, a cada palavra formada, é feito o teste novamente contra a tabela de busca.

Quando uma palavra da seqüência é encontrada na tabela de busca alcançando os parâmetros de pontuação determinados, e existem coincidências na seqüência de consulta, ocorre o que se chama de *hit*. A partir deste *hit*, ou de 2 *hits* na mesma diagonal [BKY03], dependendo também da configuração do sistema, é chamada a função que dá inicio ao terceiro passo, a extensão do *hit*.

Na função *BlastWordExtend_prelim* é testada a extensão dos *hits*. Ao concluir a extensão, a função *BlastSaveCurrentHsp* é chamada para incluir o *hit* encontrado na lista dos pares de segmentos de alta pontuação [LC00]. Finalmente, esta lista de pares de segmentos será acessada para gerar o relatório de execução do programa.

Na Figura 3 é mostrada uma árvore que representa as chamadas entre as funções do programa BlastP durante a execução do cenário de teste.

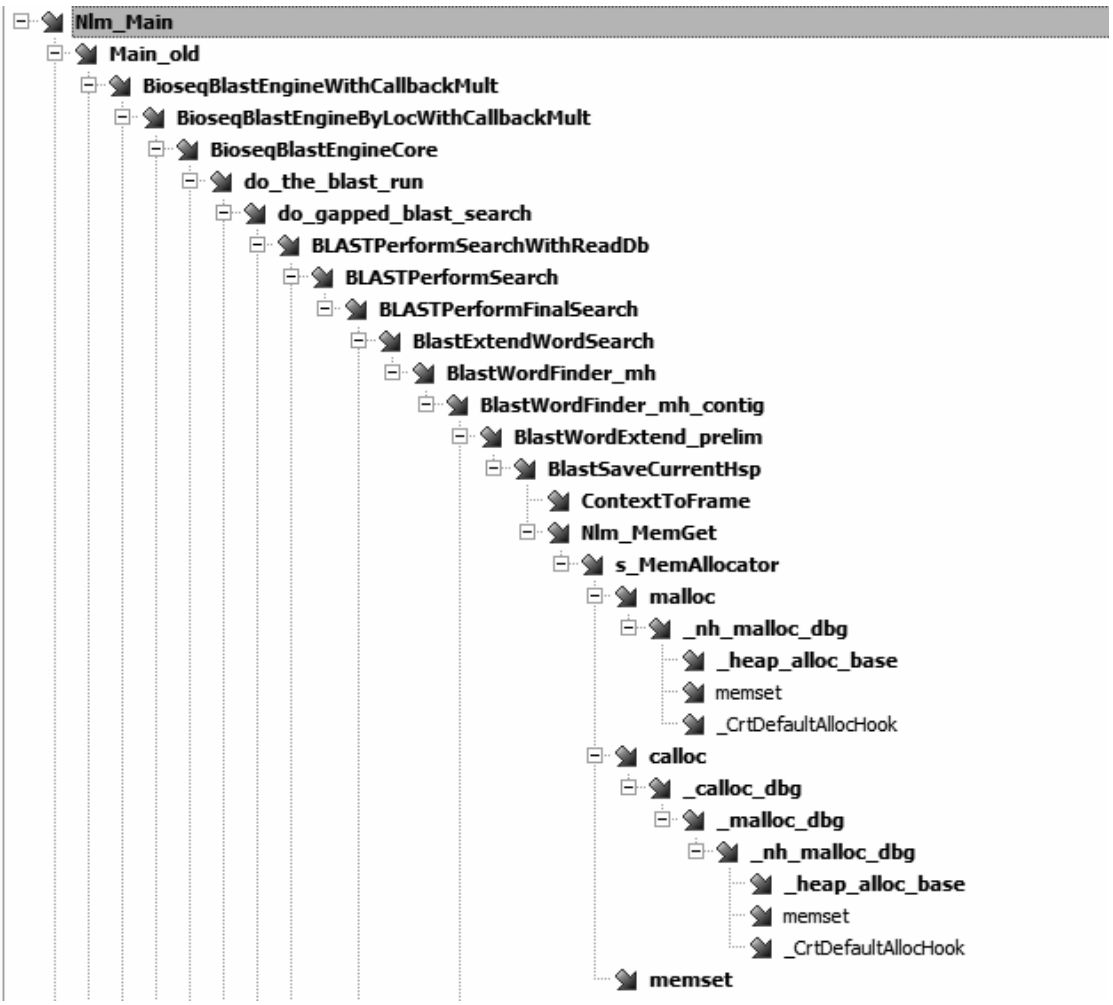

Figura 3 - Árvore de chamada das funções do BLAST.

3.3 Análise do desempenho

Esta seção detalha os resultados obtidos durante a análise de desempenho do programa BLAST. Para melhor entendimento dos resultados, foi delineada também a metodologia de análise, descrevendo o processo adotado e as ferramentas utilizadas.

3.3.1 Metodologia de análise

O processo de análise do desempenho do BLAST foi realizado com as seguintes abordagens:

1. Depuração passo a passo do código usando como ambiente de desenvolvimento o Microsoft Visual C++.

2. Introdução de rotinas de *log* para gerar registros das principais operações como criação de algumas estruturas de dados em memória e, principalmente, das operações de leitura/escritura.

3. Uso de ferramentas de *software* auxiliares para medir parâmetros como o número de operações de E/S física e tempo total de execução de cada função.

Vale ressaltar ainda as consultas feitas ao utilitário *Windows Task Manager,* disponibilizado como parte integrante do sistema operacional, que mostra informação em tempo real de uma aplicação em execução. Para as execuções foram escolhidas configurações aleatórias para o programa **BlastP**, com valores padrão do parâmetros de entrada e diferentes seqüências de consulta e diferentes fontes de dados de biosseqüências. Essas fontes de dados foram obtidas no *site* do NCBI [Ncb05b].

Na execução do programa BLAST foi utilizada a versão disponibilizada em dezembro de 2005 do *NCBI Software Development Toolkit*, versão 6.1, que contém a versão 2.0.9 do NCBI BLAST para sistema operacional Microsoft Windows.

Entre as ferramentas utilizadas estão:

- **PFMON** (*Page Faults Monitor*)[PFM06]: esta ferramenta é parte do *Resource Kit* do Windows 2000 disponibilizado pela Microsoft. É usada para medir número de operações de leitura/escritura que ocorrem durante a execução de uma aplicação. De acordo a configuração dos parâmetros de execução da ferramenta, são mostradas diferentes estatísticas sobre as operações de E/S, sendo detalhado o nome da função ou procedimento que gerou a requisição de leitura ou escritura e o tipo, *hard* ou *soft*. No primeiro caso, correspondem a E/S físicos, ou seja, acesso a disco, e o segundo, trocas de páginas de memória sem necessidade de acesso a disco. No resumo gerado pela ferramenta são apresentados totais de operações de E/S *soft*, *hard*, por acesso a dados ou do programa, entre outros.

Abaixo é mostrado na Figura 4 um exemplo de saída do PFMon. Os resultados podem ser armazenados em um arquivo plano. O detalhe das operações de E/S não é exibido na tela porque entre os parâmetros de entrada do PFMon foi escolhido "-n" que grava tal informação em um arquivo texto.

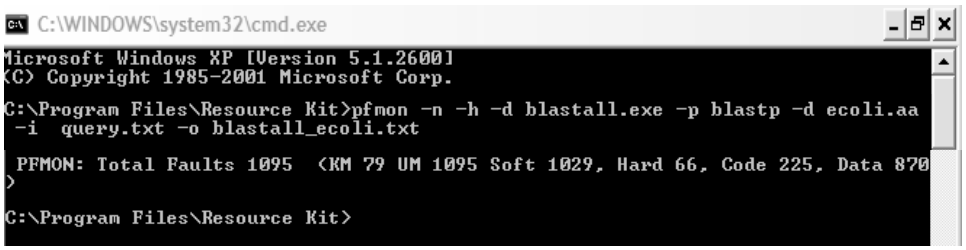

Figura 4 - Execução do PFMon para analisar page faults do BLAST

- **FileMon** (*File Monitor*)[FIL06]: ferramenta para monitorar o acesso a arquivos em tempo real. *Filemon* permite visualizar como os arquivos vão sendo utilizados por uma aplicação, mostrando quando um arquivo é aberto, ou uma operação de leitura ou escritura é requerida sobre o arquivo, até o fechamento do mesmo. Para cada operação de leitura/escritura são mostrados a posição relativa do arquivo e o número de *bytes* lidos ou escritos.

O utilitário funciona como um filtro no sistema de arquivos do Windows capturando todas as solicitações de leitura/escritura diretamente sobre os *drivers*.

Na Figura 5 abaixo é mostrada a informação gerada ao habilitar o programa FileMon para uma execução do BLAST. Estes resultados podem ser armazenados em arquivo texto.

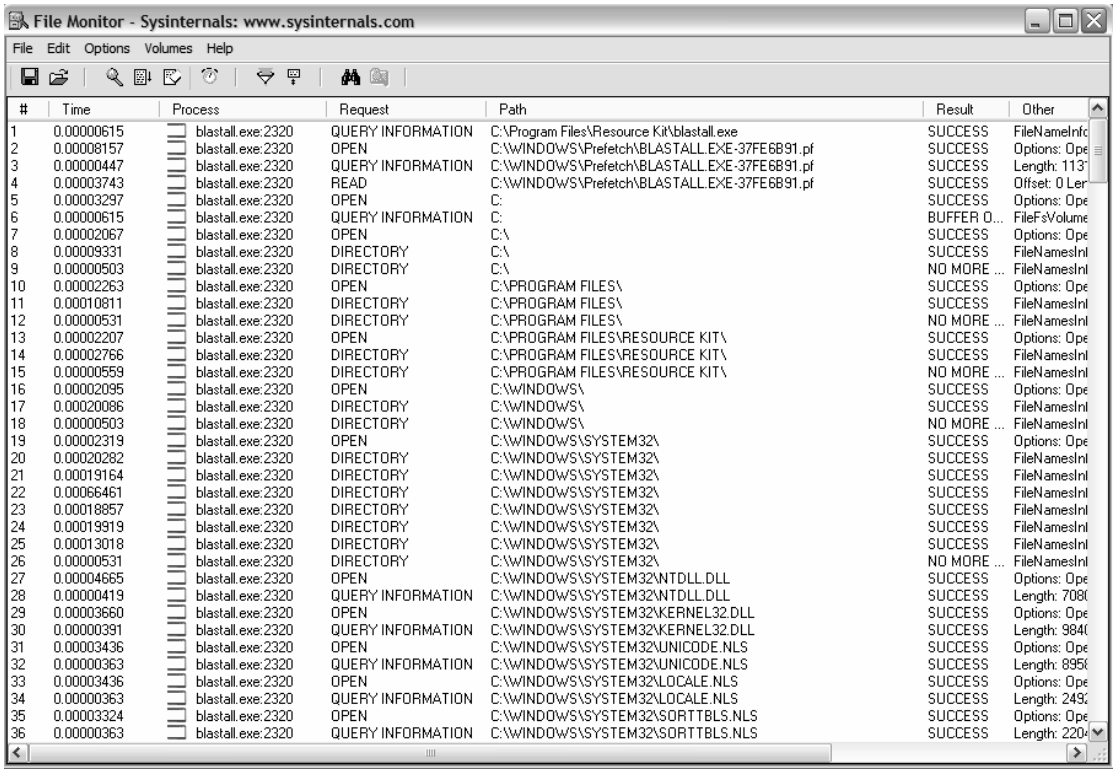

Figura 5 - Exemplo da informação captada com o utilitário FileMon.

-**AqTime** [AQT06]: esta ferramenta é similar ao PFMON, porém com uma interface gráfica que permite ver as chamadas entre as funções e analisar diversos parâmetros como número de operações de E/S, alocação de memória, tempo de execução, etc, sendo possível visualizar os resultados por função ou por linha de código. Isto é possível, pois este aplicativo analisa o programa a partir do ambiente de desenvolvimento.

Na Figura 6 é possível ver a interface principal de execução da ferramenta. Além de tabelas com resultados detalhados do parâmetro examinado durante a execução do programa, pode-se ter acesso também a árvores de execução com as chamadas às funções e resumo com as características do sistema como, memória disponível, velocidade do processador e totalizadores distintos.

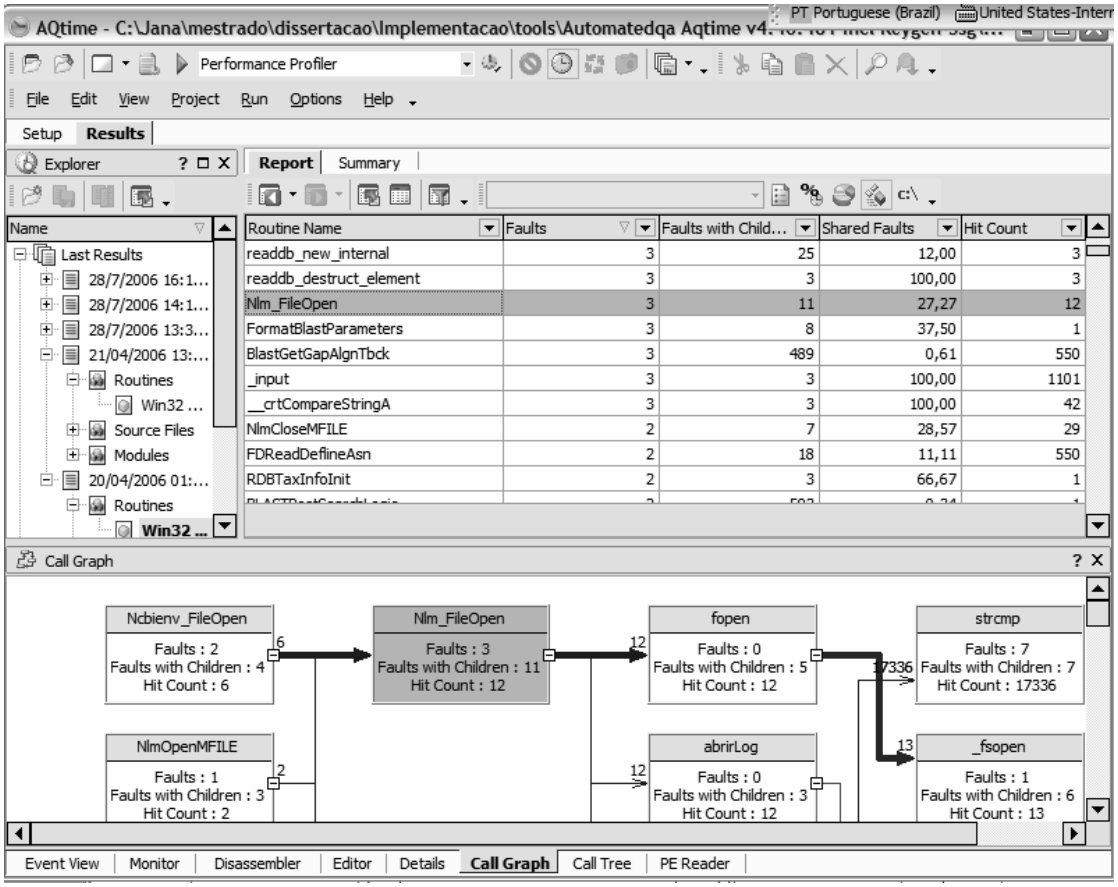

Figura 6 - Interface do aplicativo AQTime

3.3.2 Análise

Como um dos objetivos do trabalho é entender e quantificar o número de operações de E/S realizadas pelo BLAST, e dado que o programa segue as três fases bem definidas citadas na seção 3.1, as medições foram feitas tendo em conta esses referenciais, ou seja, as fases do BLAST *versus* acesso aos arquivos.

Os arquivos gerados pelo aplicativo FORMATDB, arquivo de índices (.pin), de cabeçalho (.phr) e o de seqüências (.psq), são abertos logo no inicio do programa para serem validados e também para preencher alguns campos da estrutura que guarda a configuração dos parâmetros de entrada e da estrutura principal que guarda informações de toda a execução do programa. Neste momento é lido o primeiro caractere do arquivo de seqüências, e dependendo do tamanho e quantidade de memória RAM da máquina, os arquivos de índices e cabeçalhos já são completamente carregados em memória.

Posteriormente, no inicio da primeira fase, é acessado inteiramente o arquivo com a seqüência de consulta em forma seqüencial para associar as posições do arquivo que correspondem a inicio de palavras na tabela de busca (lookup table), como mostrado na parte esquerda da figura Figura 7 [CWC06]. Ou seja, é montada uma lista de palavras a partir da seqüência de entrada, e atualizada a tabela de busca cada vez que uma palavra da entrada, ao ser comparada contra uma palavra da tabela, alcança ou supera a pontuação mínima determinada por outro parâmetro de entrada T, já que no caso das proteínas (BlastP) os casamentos inexatos também são considerados [LC00].

Na segunda fase, durante a detecção de *hits*, são acessadas uma a uma as seqüências do arquivo de seqüências, e comparado também contra a *lookup table*. Ao encontrar um casamento entre uma palavra do banco de seqüências e da *lookup*, é verificado se existe entrada correspondente à seqüência de consulta. Deste modo, nesta fase o arquivo de seqüências é completamente lido em forma seqüencial. Vale ressaltar que a estrutura da tabela de busca é intensamente acessada também. Esta comparação é ilustrada na parte mais à esquerda da Figura 7 abaixo.

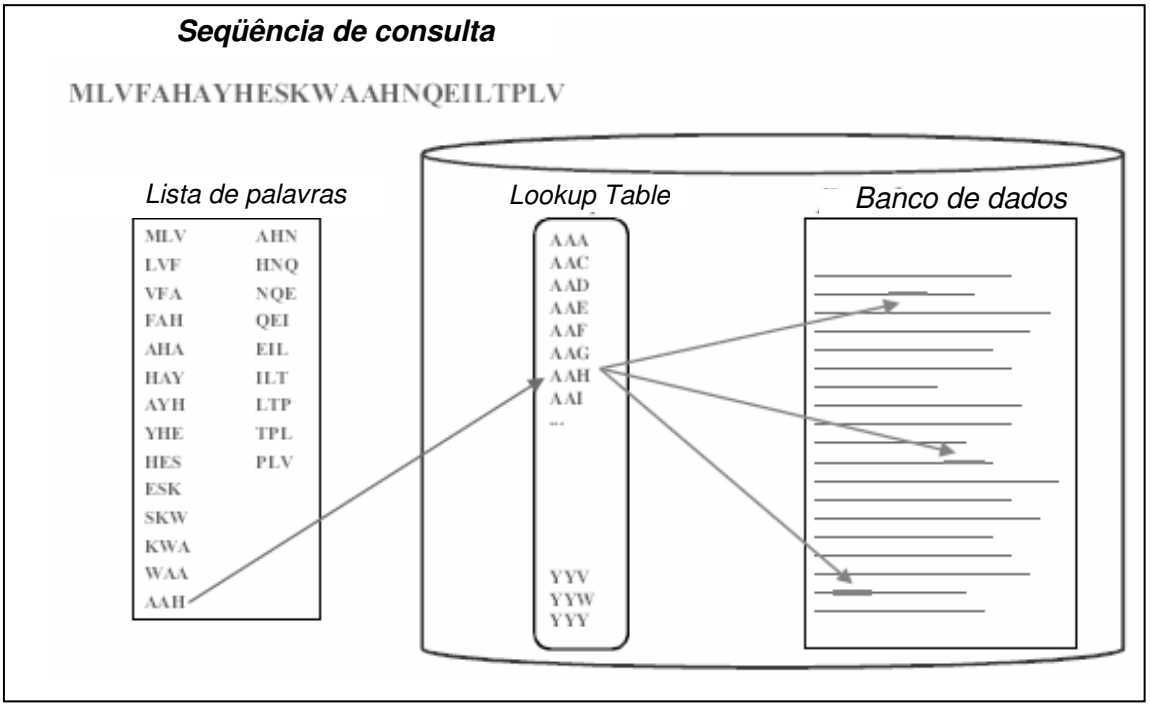

Figura 7 - Exemplo de funcionamento da primeira e segunda fase do BLAST.

Na terceira fase, apesar do acesso aos dados poder ser feito em ambas direções, direita e esquerda, geralmente só são observadas requisições aleatórias ao arquivo de seqüências para bancos muito grandes. Este comportamento é alcançado em conseqüência do recurso *memory mapped files* como também pelo modo de leitura do sistema operacional que em cada leitura traz, além dos dados requeridos, mais dados consecutivos.

Na montagem do relatório final, tanto o arquivo de cabeçalho quanto o de seqüências são requeridos. Também o arquivo de índices é usado para resolver a posição de inicio de seqüência e início de cabeçalho da seqüência procurada, ou seja, a seqüência que apresentou regiões similares à seqüência de consulta. O arquivo de seqüências nesta fase é acessado de forma aleatória, de acordo com os resultados da fase anterior, para montar os alinhamentos mostrados no relatório.

Durante a análise do programa não foi possível observar o acesso ao arquivo de seqüências na terceira fase de modo aleatório em alguns casos, como com os dados mostrados na Tabela 4, já que os arquivos eram de tamanhos pequenos em relação à quantidade de memória principal disponível. A Figura 8 apresenta um gráfico que mostra esta situação, na qual o arquivo de seqüências é acessado somente em forma seqüencial durante toda a execução do BLAST. Os dados usados para gerar o gráfico foram captados pelo aplicativo *Filemon*. O gráfico mostra o *offset*, ou posição do arquivo que foi requerida em uma operação de leitura, no eixo X, sempre crescente, e no eixo Y, a evolução do tempo. Em cada operação de leitura 32.768 bytes eram transferidos.

Dado	**Nome**	**Tamanho**
Seqüência de consulta	YRAGPRYIQAQVGPIGPRGPPGPPGSPGQQGYQG LRGEPGDSGPMGPIGKRGPPGPAGIAGKSGDDGR DGEPGPRGGIGPMGPRGAGGMPGMPGPKGHRG	100 caracteres. Porção da base Swissprot.
Banco de seqüências	Ecoli.aa (ecoli.aa.psq, ecoli.aa.pin, ecoli.aa.phr)	1.3 MB, 34 KB, 647KB.

Tabela 4 - Dados de entrada da execução do BLAST.

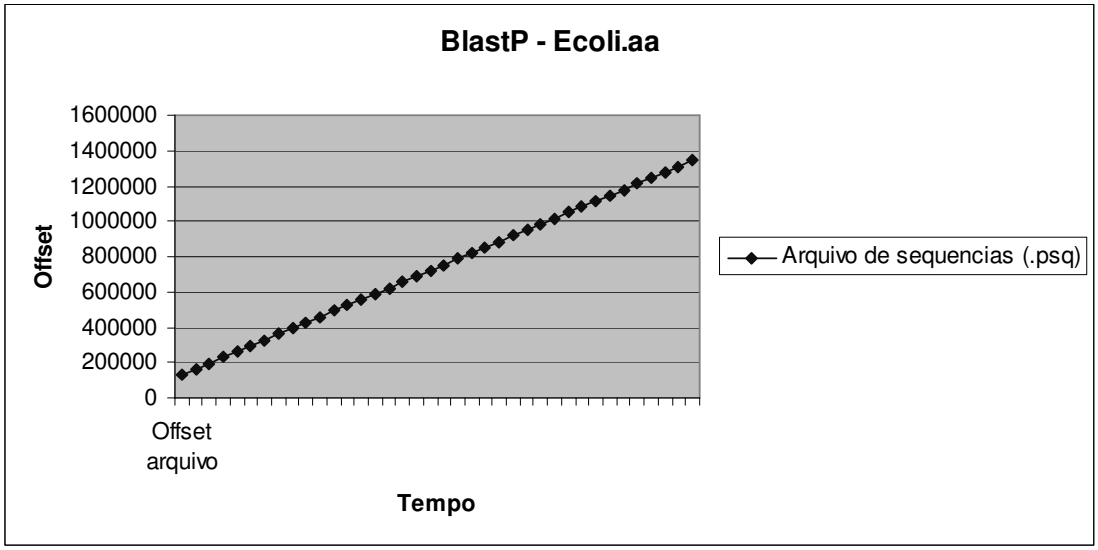

Figura 8 - Padrão de acesso ao arquivo de seqüências em uma execução do BLAST.

Já quando a comparação é feita contra um banco de seqüências muito grande, o acesso aos arquivos segue o comportamento mostrado na Figura 9, no qual é possível observar que o programa realizou leituras em forma aleatória após a leitura seqüencial do arquivo contendo as seqüências. Este gráfico foi gerado com resultados captados pelo aplicativo *FileMon* durante uma execução do BLAST com a seqüência de entrada selecionada aleatoriamente a partir da seqüência ecoli.aa contra o banco de seqüências *nr*, resumidos na Tabela 5.

Dado	**Nome**	**Tamanho**
Seqüência de consulta	MSYQEAMELSAEARAILPPENADLKTLVGNGYVVITLGLE IGADDYITKPFNPRELTIRARNLGSMPSQPDLTIIESSSQIA YKLSGRVASLHVPAVVSSARLAMELQAEPIIRSNFYRGVI HPDTDPILLTLIDTLAGDGFGKLAPS	Seqüência aleatória. 146 bases.
Banco de seqüências	nr (nr_00.psq, nr_00.pin, nr_00.phr, nr.pal)	953MB, 21.9MB, 842MB, 1KB

Tabela 5 - Dados de entrada da execução do BLAST

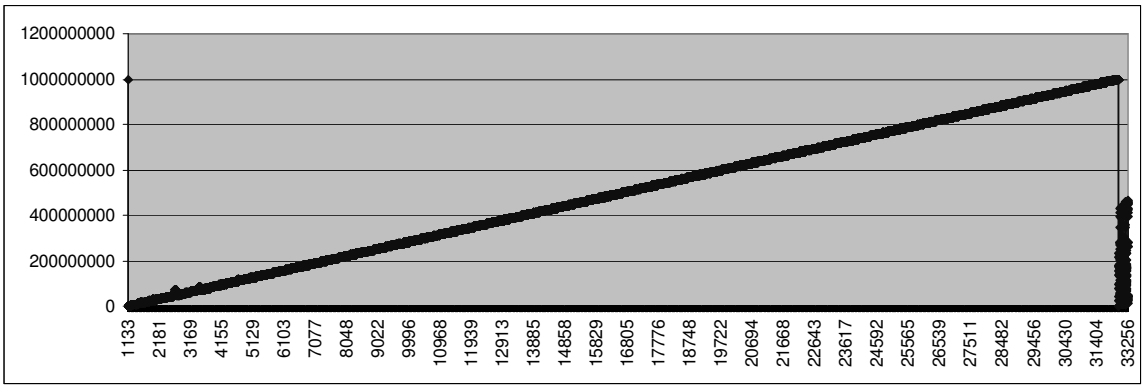

Figura 9 - Padrão de acesso ao arquivo de seqüências em uma execução do BLAST

Na Figura 10 seguinte é mostrado o detalhe do acesso aleatório ao arquivo de seqüências. Como eixo X do gráfico também foi escolhido o *offset* e como eixo Y, o número de ordem de cada operação no tempo.

Analisando os dados coletados e o gráfico, foi constatado que nesta fase de leitura randômica, os *offsets* não se repetiram como se esperava, sendo a primeira requisição correspondendo à terceira fase do algoritmo, e para algumas posições, no caso de ter encontrado um alinhamento com sucesso, para montagem do relatório. De aproximadamente 8000 requisições de leitura randômica somente 37 posições foram acessadas duas vezes, sendo que uma vez durante a leitura seqüencial e a outra no momento de leitura aleatória. Isto ocorre, pois a extensão de *hits*, terceira fase, é testada logo que um *hit* é encontrado, e não iniciada uma vez concluída a segunda etapa para todo o banco de seqüências.

Esta afirmação foi confirmada pelo estudo do código aqui realizado, identificando o momento da chamada da função que testa a extensão. Portanto, estes acessos aleatórios correspondem à etapa de elaboração de relatório, no qual as seqüências são lidas para montar o alinhamento com a seqüência de entrada.

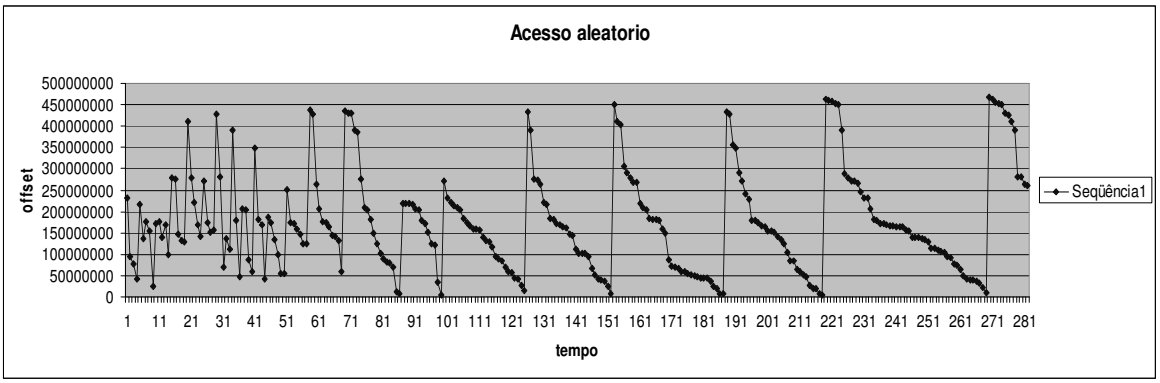

Figura 10 - Acesso aleatório ao arquivo de seqüências em uma execução do BLAST.

Do ponto de vista do desempenho do BLAST, foram realizadas algumas medições para obter o número de operações de E/S requeridas e tempo de execução. Na Tabela 6 é possível verificar o número total de operações E/S, (*hard page fault*) por função, coletado com o aplicativo PFMon. Como se pode observar, apenas cinco funções representam 97% das operações de leitura realizada pelo programa, sendo que 83% são realizadas por uma única função.

Função	Page Fault	Percentual
BlastWordFinder_mh_contig	3.885.723	83,15%
BlastWordExtend_prelim	553.110	11,84%
readdb_get_sequence	49.112	1,05%
Memcpy	43.289	0,93%
TOTAL	**4.672.962**	**97%**

Tabela 6 - Resultados da execução do BLAST com a base *nr* captados pelo PFMon.

Na Tabela 7 é possível visualizar um resumo dos dados coletados com o AQTime executando o BLAST com uma seqüência de consulta aleatória contra o banco env_nr. Foi possível verificar quais funções ou procedimentos realizavam o maior núumero de operações. Comparando os resultados captados pelos dois aplicativos, mesmo para cenários de teste diferentes, são confirmadas as funções mais caras do BLAST em relação às operações de E/S.

Função	Page Faults	Percentual	Arquivo
BlastWordFinder_mh_contig	5.652	83,28%	blast.c
NlmReadMFILE	506	7,46%	readdb.c
BlastWordExtend_prelim	244	3,60%	blast.c
readdb_get_defline_ex	133	1,96%	readdb.c
readdb_get_sequence	122	1,80%	readdb.c
lookup_find_init	54	0,80%	lookup.c
TOTAL	**6.711**	**98,90%**	

Tabela 7 - Resultados da execução do BLAST com a base env_nr captados pelo AQTime.

Em outra execução do BLAST monitorada pelo aplicativo AQTime, focando a análise em cada linha de código, foi possível verificar, dentro da função que mais realizou *page faults*, quais linhas contribuíam com a maioria de tais requisições. Como resultado constatou-se que uma única linha é responsável por 90% das operações de leitura. Esta linha de código é mostrada isoladamente no Figura 11. Nela mostra o consumo de dados de uma seqüência do banco de seqüências. Nesta linha de código a seqüência em questão está sendo comparada contra a seqüência de consulta. A seqüência que está sendo processada é representada pela variável S. As operações de bits e mascaramento são usadas para obterem-se os *bits* mais significativos da representação de cada *byte* que representa uma base. A operação (s+1) é onde se dispara as requisições de dados. Neste momento do programa a seqüência é lida em forma seqüencial.

```
static Int4 BlastWordFinder_mh_contig(BlastSearchBlkPtr search,
LookupTablePtr lookup)
{  ...
       next_lindex = (((lookup_index) & mask)<<char_size) + *(s+1);
   ...
}
```

Figura 11 - Linha de código onde são acessados os dados de uma Biosseqüências

3.4 Conclusão

Uma vez entendido os passos realizados pelo programa BlastP durante a comparação de seqüências para busca de similaridades e obtido resultados do

comportamento do mesmo em vários cenários de execução em relação as operações de leitura de dados em memória secundaria, atinge-se um problema concreto a ser resolvido: para reduzir o numero de operações de E/S realizadas pelo BLAST deve-se atacar a segunda fase do algoritmo. Do ponto de vista da implementação isto corresponde a atacar a função *BlastWordFinder_mh_contig*, que realiza leituras seqüenciais sobre os dados.

Uma proposta para resolver este problema é fazer uso de técnicas de compactação de dados de maneira a reduzir o volume de dados e conseqüentemente poder transferir para a memória principal mais dados em cada operação de leitura.

Desta maneira, no próximo capitulo serão investigados mecanismos de compactação para biosseqüências como forma de persistência para garantir um acesso mais eficiente pelo programa BlastP.