6 Resultados experimentais

Neste capítulo foram analisados os resultados obtidos após a realização dos testes efetuados sobre a proposta implementada. Neste sentido, foram definidos alguns cenários de execução interessantes para poder perceber o impacto gerado da solução proposta sobre o programa BlastP. Ao final deste capitulo são apresentadas notas conclusivas comparando o resultado inicialmente esperado com os resultados realmente obtidos.

6.1 Metodologia de análise de resultados

O primeiro passo para um bom planejamento experimental é conhecer com profundidade o problema e o sistema que se deseja analisar [BUT05]. Neste caso o ***problema*** está relacionado com a redução do tempo de latência nas operações de entrada e saída das aplicações biológicas intensivas sobre os dados de biosseqüências. O ***sistema*** que estará sendo analisado é um dos programas da família BLAST, BlastP, o qual realiza comparação por similaridade entre biosseqüências. No sentido de buscar uma solução para o problema e o sistema em questão, foram utilizadas técnicas de banco de dados em conjunto com técnicas de compactação de dados como já discutido anteriormente.

Sendo assim, serão estabelecidos os critérios para determinar os testes necessários com o objetivo de medir a influência das variáveis introduzidas no sistema através das novas técnicas adotadas. Desta forma, nas seções subseqüentes serão detalhados:

1. Os instrumentos, no caso, as configurações de hardware e software, incluindo ferramentas para captar os resultados.

2. O processo, incluindo dados, número de testes e a seqüência dos mesmos.

3. As variáveis a serem observadas.

4. As variáveis introduzidas mediante alterações no sistema.

Configuração de Hardware e Software

A **Tabela 9** ilustra o cenário padrão de execução do BLAST. Foi utilizada uma única máquina para realizar os testes, com quantidade de memória, processador e sistema operacional mostrado abaixo.

Memória total (RAM)	1 GB
Memória disponível (RAM)	Entre 450 e 550 MB.
Processador	Intel(R) Pentium(R) M (Centrino) – 2 GHz
Sistema Operacional	Microsoft Windows XP Home Edition 2002 Service Pack 2

Tabela 9 - Configuração de Hardware e Software Básico

A coleta de dados foi realizada com os mesmos programas utilizados durante a etapa de análise de desempenho do BLAST, resumidos a seguir:

1. Ambiente de desenvolvimento, Microsoft Visual C++[MVC06].

2. Utilitário para medir número de falhas na leitura de páginas em memória as quais necessitaram acesso a disco (e.g. em inglês *hard page faults*): AQTime[AQT06].

O Processo

O processo para análise dos resultados é composto por uma série de cenários, onde o resultado de execução de cada cenário gerará um ou mais gráficos com análises explicativas sobre os mesmos. Cada cenário é representado por um conjunto de variáveis de entrada e um conjunto de variáveis de saída. A cada execução de cenários serão atribuídos valores às variáveis de entrada e coletados os valores das variáveis de saída. Por exemplo, uma importante variável de saída é o número de operações de entrada e saída executadas durante o respectivo cenário.

O objetivo desta tática é avaliar o uso de estratégia proposta em base ao comportamento do NCBI-BLAST e a correlação entre as variáveis definidas.

Os cenários foram executados usando um arranjo conhecido como cache frio, reiniciando o computador a cada execução monitorada para evitar a influencia das variáveis não controláveis, como a reutilização de dados já em memória. Este arranjo foi escolhido por representar com mais fidelidade a

realidade de um laboratório, já que não é tão comum uma situação na qual uma execução do BLAST seja feita varias vezes para uma mesma seqüência e banco, diferente da utilização de um banco de dados multi-usuários onde a mesma consulta é submetida muitas vezes ao sistema.

A **Figura 20** ilustra os componentes presentes em cada cenário de execução. Os componentes são: (1) biosseqüência de entrada, (2) programa NCBI-BLAST, (3) relatório de resultado do BLAST, (4) módulo de gerência de memória o qual implementa a nossa solução proposta e (5) fonte de dados de biosseqüência.

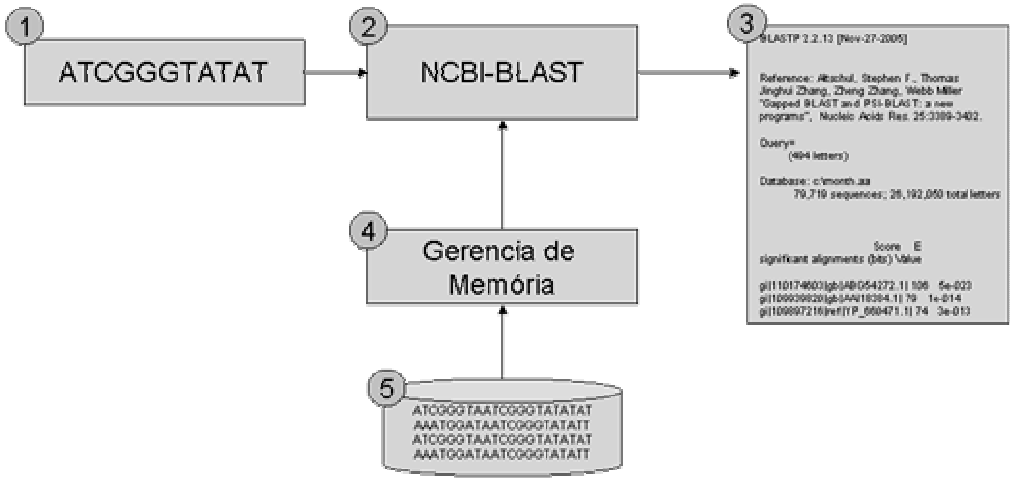

Figura 20 - Componentes do Cenário de Execução

De acordo com a Figura 20, os cenários de execução foram definidos levando em conta as seguintes variáveis:

1. **Volume de dados** dos bancos de biosseqüências mais populares;

2. **Grau de similaridade** entre a seqüência de consulta e o banco de seqüências;

3. **Tamanho da seqüência de consulta**.

Todas as fontes de dados utilizadas podem ser obtidas no NCBI via *ftp* (ftp://ftp.ncbi.nih.gov/blast/db/).

Cenário	Seqüência de consulta	Banco de seqüências	Observação
1	seqEntrada500_swissprot.txt	month.aa (25MB) Julho 2006.	Seqüência de 500 bases aleatórias extraídas do banco Swissprot. O banco month.aa contem as novas entradas registradas no banco no mês.
2	seqEntradaVarias.txt	env_nr (195 mB)	A seqüência de entrada foi criada a partir do banco de seqüências mito.aa e yeast.aa, sendo escolhidas 3 seqüências aleatoriamente.

Tabela 10 - Cenários de execução

A Tabela 10 apresenta os dois cenários investigados neste trabalho com as respectivas biosseqüências de consultas e fontes de dados de biosseqüências utilizadas. Na coluna "Observação" são apresentadas algumas informações com respeito à forma como essas seqüências foram obtidas.

Variáveis de saída observadas

A análise de desempenho do BLAST, apresentada no capítulo 3, suscitou um problema concreto a ser atacado, já que foi possível observar que a aplicação não escapa à conhecida regra "80-20", Principio de Pareto[PAR06], na qual mais de 80% das conseqüências tem origem em 20% das causas. No caso, as operações de leitura físicas foram provocadas por uma única função e sobre o arquivo de seqüências, e dentro desta, 90% em uma única linha de código.

Como apresentado no capítulo 3, foi constatado que a função ***BlastWordExtend_mh_contig*** representa quase 85% do total das operações de entrada e saída registradas. Como foi observado durante a análise da execução da função acima, a leitura é realizada de forma seqüencial. Sendo assim, uma das técnicas que se mostra adequada para reduzir o número de operações de leitura é a de compressão de dados, já que desta maneira aumenta-se a *densidade da informação* a cada transferência de disco para memória principal.

Para avaliar a sobrecarga causada pelas alterações do programa, foram observadas durante todas as execuções do BLAST as seguintes variáveis:

- Número de operações de E/S realizadas;

- Tempo de execução de cada função.

Alterações no sistema

De acordo a proposta apresentada no capítulo 5, trabalhou-se com três versões do sistema BLAST:

1. Blast original, sem alterações.

2. Blast com gerência de memória: incluindo no programa somente o módulo *GerenteMemoria* (vide capítulo 5) com o intuito de avaliar a carga deste componente separadamente da compressão em si.

3. Blast com gerência de memória e dados compactados: utilização da arquitetura completa proposta no capítulo 5.

Nesta terceira versão, com o arquivo compactado em blocos de tamanho fixo, foram testadas as abordagens:

a) Ingênua: descompacta totalmente antes do primeiro acesso ao arquivo.

b) Em blocos: a descompactação ocorre sob demanda, ou seja, a cada requisição de leitura é verificado se o dado se encontra em memória ou não. Em caso negativo, é disparada a função que descompacta mais um bloco.

A escolha destas abordagens foi baseada no estudo da aplicação de técnicas de compactação a banco de dados, detalhadas no capítulo 3.

Foram testados três tamanhos de bloco, 32k, 256k e 512k, dado que tamanhos diferentes resultam em taxas de compactação diferentes, escolhidos aleatoriamente, porém levando em consideração o tamanho de bloco de transferência de dados utilizado pelo sistema operacional.

A Tabela 11 resume o planejamento experimental apresentando os 10 testes que serão realizados. A sigla usada em cada cabeçalho de célula da tabela representa respectivamente:

- BLAST: refere-se ao uso do programa NCBI-BLAST original sem alterações;
- BLASTgm: onde será utilizado o NCBI-BLAST com gerencia de memória, porém com os dados originais sem compactação;
- BLASTc: faz uso do NCBI-BLAST com estratégia de compressão ingênua;
- BLASTcBlk: se refere ao uso do NCBI-BLAST com estratégia de compressão e descompressão por blocos sobre demanda.

Programa / Cenário	BLAST	BLASTgm	BLASTc			BLASTcBlk		
			32k	256K	512K	32k	256k	512k
Cenário 1	T1	T2	X	T3	X	T4	T5	X
Cenário 2	T6	T7	T8	X	T9	X	X	T10

Tabela 11 - Planejamento experimental

Cada célula da tabela apresenta um código que identificará o teste a ser realizado. Desta maneira, por exemplo, o código T8 representa o teste usando o NCBI-BLAST com estratégia ingênua de compactação usando a seqüência de consulta seqEntradaVarias.txt e a fonte de dados env_nr. Alguns testes, marcados com X na tabela, não serão realizados por considerar-se que não agregaria mais informação sobre o desempenho, já que estará representado por um teste equivalente, porém com outra entrada.

Vale ressaltar que outros cenários também foram testados, porém somente serão relatados aqui somente os testes mostrados na Tabela 11.

6.2 Análise de resultados

Nesta seção serão apresentados e analisados os resultados obtidos durante a experimentação. Serão utilizadas tabelas para mostrar os valores coletados para as duas variáveis observadas em cada teste. Os dados foram coletados com o aplicativo AQTime.

A análise será feita para cada entrada, seqüência de consulta e banco de seqüências, correspondente a uma linha da tabela de Planejamento Experimental mostrada seção 6.2, comparando o desempenho de cada versão do programa.

6.2.1 Resultados e Análises do Cenário 1- Testes de T1 a T5

Nesta seção apresentaremos o primeiro conjunto de testes realizado no contexto do Cenário 1. A configuração deste cenário é resumida na **Tabela 12**.

Seqüência de consulta	seqEntrada500_swissprot.txt
Banco de Seqüências	Month.aa
Programa	Blastp
Tamanho do arquivo .psq	26.271.770 bytes
Tamanho do arquivo .psq compactado – bloco 32kb	13.827.780 bytes
Tamanho do arquivo .psq compactado- bloco 256kb	13.771.078 bytes

Tabela 12 - Configuração do testes T1 a T5

Os resultados obtidos nas execuções do primeiro cenário para a variável observada número de operações de E/S, ou *hard page faults,* são resumidos na Figura 21.

Observando o gráfico é possível comprovar que o objetivo foi alcançado, ou seja, o número de operações de E/S foi reduzido quase pela metade. Mais exatamente em 43% entre a versão original e a versão compactada com maior redução, como pode ser visto na Tabela 13 mais abaixo.

Neste cenário verificou-se que entre as versões com compactação, a versão que descompacta o arquivo todo de uma única vez, BLASTc, realiza o menor número de operações de E/S. Vale ressaltar que a versão denominada BLASTc descompacta os dados em uma única chamada, porém o algoritmo de compactação é aplicado em blocos e não sobre o arquivo total, já que a abordagem de compactar o arquivo completo resultou em tempos demasiadamente superiores ao BLAST original, sendo por isso abandonada e os resultados não incluídos no trabalho. Inclusive, em alguns casos nem era

possível concluir a execução, pois as estruturas usadas durante a compactação não conseguiam alocar memória para armazenar os resultados intermediários.

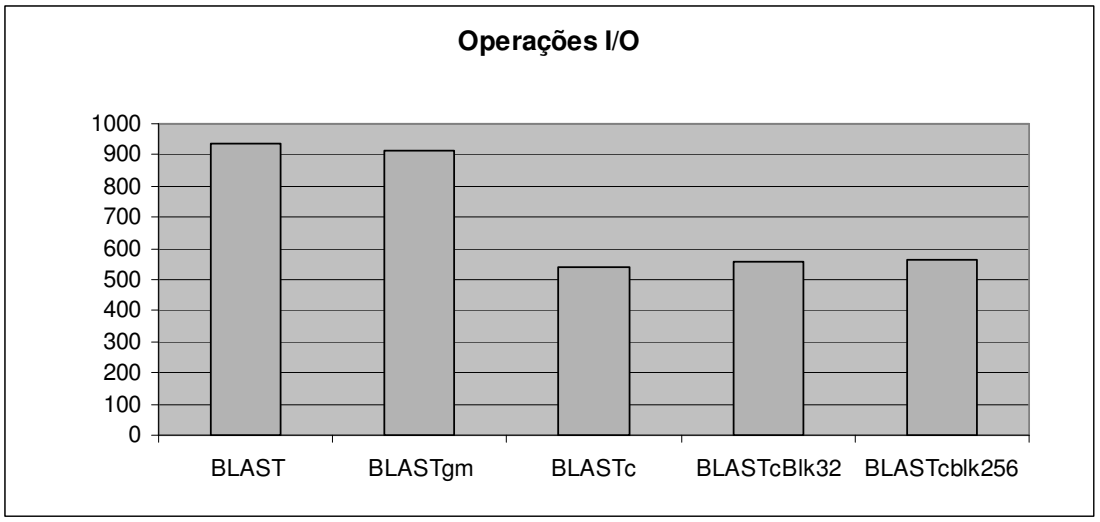

Figura 21 - Número de operações de E/S para as execuções do cenário

Por outro lado, como pode ser visualizado na Figura 22, apesar de obter um desempenho melhor em relação ao numero de operações de E/S, a versão BLASTc impôs uma sobrecarga de processamento maior que nas outras soluções. Ao observar o detalhe das medições por função nas figuras abaixo é ainda mais claro que na versão BLASTc a função que finalmente descompacta consome mais tempo que nas demais versões que tratam os dados compactados, é como se o custo de descompactação fosse diluído no processamento total do BLAST.

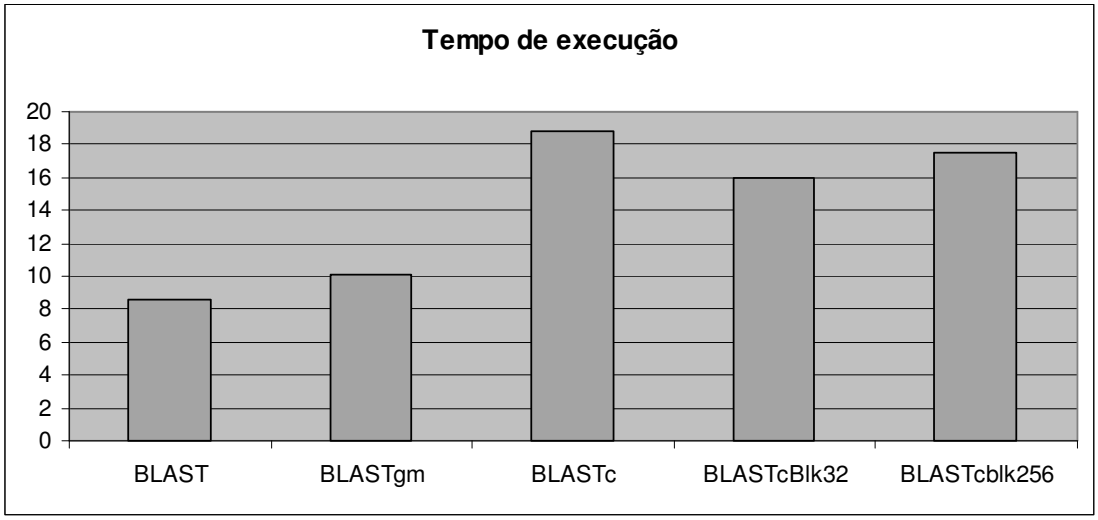

Figura 22 - Tempo de execução total para as execuções do cenário 1

Para as duas variáveis observadas, tempo de execução e número de operações de E/S, ou número de falhas de acesso aos dados requerendo acesso a disco (*hard page faults)*, são apresentados na Tabela 13.

Programa	Tempo de execução (em segundos)	Hard page Faults
BLAST	8,58	934
BLASTgm	10,07	911
BLASTc	18,79	539
BLASTcBlk32	16,02	558
BLASTcblk256	17,46	563

Tabela 13 - Resumo dos resultados para o primeiro cenário

A seguir são apresentadas tabelas com o custo de operações de E/S e tempo de execução detalhado por função ou procedimento do programa BLAST.

- T1: execução do BLAST original.

Na Tabela 14 são corroborados os dados apresentados no capítulo 3 referente ao custo da função BlastWordFinder_mh_contig. Neste teste, tal função concentra o maior número de operações de E/S de todo o programa e também consome maior percentual de tempo de execução.

Função	Page Faults	Percentual
BlastWordFinder_mh_contig	669	77,3%
BlastWordExtend_prelim	127	14,7%
NlmReadMFILE	39	4,5%
AsnModuleLink	18	2,1%
_sopen	13	1,5%

Função	Tempo	Percentual
BlastWordFinder_mh_contig	3,61	42,1%
BlastWordExtend_prelim	0,92	10,7%
write_char	0,9	10,5%
_output	0,84	9,8%
SEMI_G_ALIGN_EX	0,65	7,6%

Tabela 14 - Resultados do teste T1.

- T2: execução do BLAST com gerente de memória (BLASTgm).

Como esta versão somente incorpora o gerente de memória, o qual verifica se um determinado dado já está em memória antes de requerer uma operação de descompactação, é esperado que a função correspondente ao gerente agregue um custo de processamento, como pode ser visto na Tabela 15. Porém, esta função acaba não realizando requisições de dados, pois isto continua sendo feito através do recurso *Memora Mapped Files*, e as funções que mais geram *hard page faults* continuam as mesmas e praticamente na mesma proporção.

Função	Page Faults	Percentual
BlastWordFinder_mh_contig	669	73,4%
BlastWordExtend_prelim	127	13,9%
NlmReadMFILE	39	4,3%
AsnModuleLink	17	1,9%
AddXMLname	11	1,2%
readdb_get_sequence	10	1,1%

Função	Tempo	Percentual
solicitaBlocoDescomp	3,98	39,5%
BlastWordFinder_mh_contig	3,52	35,0%
BlastWordExtend_prelim	1,27	12,6%
SEMI_G_ALIGN_EX	0,48	4,8%
readdb_get_sequence	0,14	1,4%

Tabela 15 - Resultados do teste T2.

- T3: Execução da versão do BLAST com descompactação total (BLASTc).

Com este teste já é possível ver a sobrecarga da descompactação sobre a execução do BLAST. A direita na figura observa-se que a tarefa da leitura foi realizada na função que decodifica os dados (*huffman_decode_memory*), e não mais com a função correspondente ao gerente de memória (*solicitaBlocodescomp*). Outras funções do procedimento de descompactação,

como *UnMTF* e *UnBWT*, também aparecem na tabela entre as que mais consumiram tempo de processamento.

Este resultado deve-se em parte a que o banco de seqüências é relativamente pequeno em relação à quantidade de memória principal disponível. Assim, uma vez descompactado, as próximas funções terão acesso aos dados de forma mais rápida, tornando as funções de descompactação relativamente mais caras que as demais funções.

Os resultados são mostrados na Tabela 16.

Função	Page Faults	Percentual	Função	Tempo	Percentual
huffman_decode_memory	420	77,9%	huffman_decode_memory	6,23	33,2%
NlmReadMFILE	41	7,6%	solicitaBlocoDescomp	4,02	21,4%
AsnModuleLink	17	3,2%	UnMTF	2,43	13,0%
AddXMLname	11	2,0%	BlastWordFinder_mh_contig	2,11	11,2%
readdb_get_sequence	10	1,9%	UnBWT	1,22	6,5%
			BlastWordExtend_prelim	1,07	5,7%

Tabela 16 - Resultados do teste T3.

- T4: BLAST com compactação – descompactação em bloco (bloco = 32k).

Este teste apesar de obter resultados parecidos ao teste anterior, apresenta um maior equilíbrio entre a função que solicita a descompactação de um novo bloco e as funções que descompactam. Os detalhes são mostrados na Tabela 17.

Função	Page Faults	Percentual	Função	Tempo	Percentual
huffman_decode_memory	418	74,9%	solicitaBlocoDescomp	3,96	24,7%
NlmReadMFILE	39	7,0%	huffman_decode_memory	3,58	22,3%
AsnModuleLink	18	3,2%	UnMTF	2,37	14,8%
AddXMLname	11	2,0%	BlastWordFinder_mh_contig	2,1	13,1%
readdb_get_sequence	10	1,8%	UnBWT	1,07	6,7%
			BlastWordExtend_prelim	1,03	6,4%

Tabela 17 - Resultados do teste T4.

- T5: Execução do BLAST com descompactação em bloco (bloco = 256k).

Na Tabela 18 abaixo são mostrados os resultados do teste T5. Comparando os dados obtidos com o teste anterior, teste T4, o teste T5 mantém o mesmo padrão de comportamento em termos percentuais, mas em valores absolutos, pode-se ver que o tempo de execução da função de decodificação (*huffman_decode_memory*) é maior, já que o tamanho do bloco a descompactar é maior.

Função	Page Faults	Percentual
huffman_decode_memory	418	74,2%
NlmReadMFILE	39	6,9%
AsnModuleLink	18	3,2%
AddXMLname	11	2,0%
readdb_get_sequence	10	1,8%

Função	Tempo	Percentual
huffman_decode_memory	5,05	28,9%
solicitaBlocoDescomp	3,99	22,9%
UnMTF	2,39	13,7%
BlastWordFinder_mh_contig	2,13	12,2%
UnBWT	1,23	7,0%
BlastWordExtend_prelim	1,06	6,1%

Tabela 18 - Resultados do teste T5.

6.2.2 Resultados e Análises do Cenário 2 – Testes de T6 a T10

Nesta seção apresentaremos o segundo conjunto de testes realizado no contexto do Cenário 2. A configuração deste cenário é resumida na Tabela 19 mostrada abaixo.

Seqüência de consulta	seqEntradaVarias.txt
Banco de Seqüências	Env_nr
Programa	Blastp
Tamanho do arquivo .psq	195.016.681 bytes
Tamanho do arquivo .psq compactado – bloco 32kb	106.083.827 bytes
Tamanho do arquivo .psq compactado- bloco 512kb	105.163.287 bytes

Tabela 19 - Configuração dos testes T6 à T10

O gráfico mostrado na Figura 23 permite visualizar a comparação entre os resultados de cada execução para o número de *hard page faults*. Mais uma

vez foi reduzido o número de operações de E/S nas versões alteradas do programa. Porém, neste cenário a versão que obteve melhor desempenho foi a denominada blastc512, a qual descompacta em uma única chamada à função de descompactação todo o arquivo de seqüências compactado em blocos de 512K. Esta configuração não havia sido testada no cenário anterior, somente a versão blastc32, onde se segue o mesmo procedimento, porém o tamanho do bloco é menor, mas, de todas as maneiras os resultados seguiram a mesma linha dos testes anteriores, ou seja, a descompactação em uma única chamada terminou realizando um menor número de operações de E/S.

Também como no cenário anterior, a versão original e a versão com o gerente de memória apresentaram resultados similares.

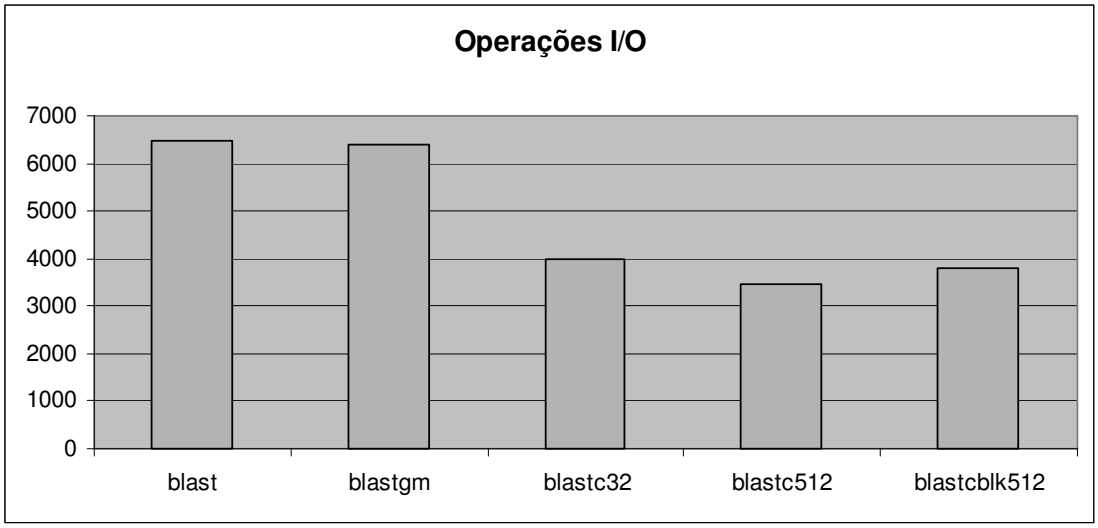

Figura 23 - Número de operações de E/S para as execuções do cenário 2.

Analisando o gráfico de tempo de execução mostrado na Figura 24, novamente o custo da descompactação fez com que as versões alteradas apresentassem tempos de execução total bem maiores que a versão original. É possível constatar que desde o uso do gerente de memória já representa uma carga considerável na execução do BlastP.

Também o tamanho do bloco utilizado para compactar os dados não só influi no grau de compactação como mostrado no capítulo 5, como também no tempo de execução do programa. Apesar de que, para blocos menores, as

funções de descompactação são chamadas mais vezes, o tempo de execução acaba sendo maior para blocos maiores.

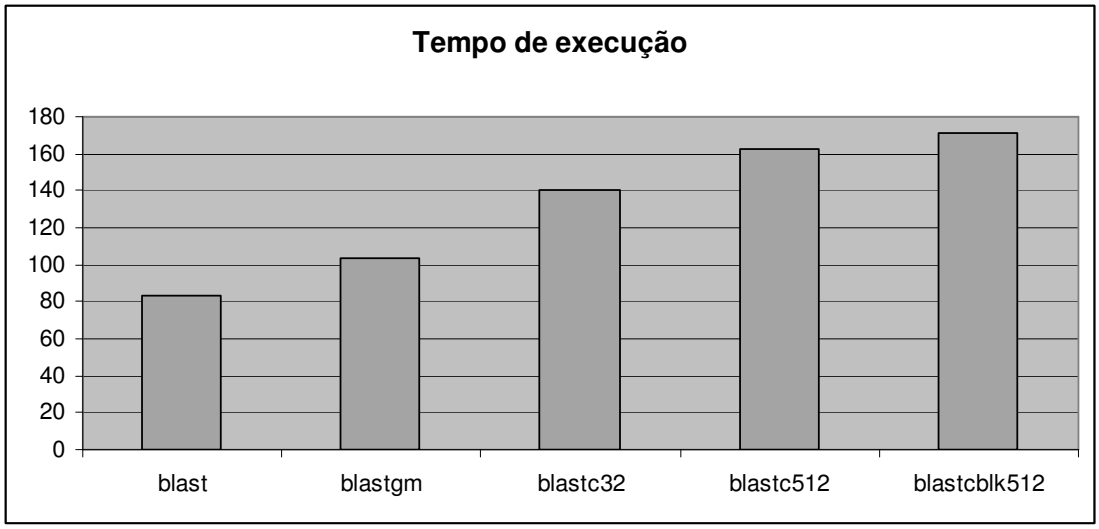

Figura 24 - Tempo total de execução para as execuções do cenário 2.

Os resultados obtidos para as duas variáveis observadas, tempo de execução e número de operações de E/S (*hard page faults*), são resumidos na Tabela 20.

Programa	Hard page Faults	Tempo de execução (em segundos)
blast	6488	83,2
blastgm	6383	103,99
blastc32	3981	140,71
blastc512	3471	162,12
blastcblk512	3804	170,93

Tabela 20 - Resultados testes T6 à T10.

A seguir são apresentadas as tabelas com dados coletados nos testes denominados T6 à T10 no mesmo formato do cenário anterior.

Os resultados obtidos neste conjunto de testes mostraram algumas diferenças de comportamento em relação ao conjunto anterior em conseqüência do volume de dados tratado aqui ser muito maior, o banco env_nr[Ncb05c] é aproximadamente sete vezes maior que o banco month.aa. Desta maneira, a

função que gerência os dados em memória principal consumiu o maior tempo de processamento em todos os testes.

Com relação ao número de operações de E/S, o destaque ficou com a função do último passo do algoritmo de descompactação, huffman_decode_memory, mesmo quando a descompactação foi realizada sobre demanda.

- T6: execução do BLAST original.

Função	Page Faults	Percentual
BlastWordFinder_mh_contig	5780	89,1%
NlmReadMFILE	255	3,9%
readdb_get_sequence	122	1,9%
BlastWordExtend_prelim	114	1,8%
readdb_get_defline_ex	101	1,6%

Função	Tempo	Percentual
BlastWordFinder_mh_contig	25,7	30,9%
write_char	19,23	23,1%
_output	18,19	21,9%
_write	6,16	7,4%
SEMI_G_ALIGN_EX	1,82	2,2%

Figura 25 - Resultados da execução do teste T6.

- T7: execução do BLAST com gerente de memória.

Função	Page Faults	Percentual
BlastWordFinder_mh_contig	5695	89,2%
NlmReadMFILE	255	4,0%
readdb_get_sequence	122	1,9%
BlastWordExtend_prelim	112	1,8%
readdb_get_defline_ex	100	1,6%

Função	Tempo	Percentual
solicitaBlocoDescomp	50,58	48,6%
BlastWordFinder_mh_contig	38,91	37,4%
BlastWordExtend_prelim	3,16	3,0%
SEMI_G_ALIGN_EX	1,83	1,8%
readdb_get_sequence	1,72	1,7%

Figura 26 - Resultados da execução do teste T7.

- T8: execução do Blastc32 que realiza descompactação total (blocos de 32k)

Função	Page Faults	Percentual
huffman_decode_memory	3221	80,9%
NlmReadMFILE	257	6,5%
readdb_get_sequence	154	3,9%
readdb_get_defline_ex	124	3,1%
AsnModuleLink	18	0,5%

Função	Tempo	Percentual
solicitaBlocoDescomp	49,95	35,5%
BlastWordFinder_mh_contig	27,27	19,4%
huffman_decode_memory	20,66	14,7%
UnMTF	17,4	12,4%
UnBWT	7,57	5,4%

Figura 27 - Resultados da execução do teste T8.

.

- T9: execução do Blastc512, que realiza descompactação total (blocos de 512k)

Função	Page Faults	Percentual	Função	Tempo	Percentual
huffman_decode_memory	2892	83,3%	solicitaBlocoDescomp	50,2	31,0%
NlmReadMFILE	256	7,4%	huffman_decode_memory	32,36	20,0%
readdb_get_sequence	133	3,8%	BlastWordFinder_mh_contig	27,6	17,0%
readdb_get_defline_ex	127	3,7%	UnBWT	17,77	11,0%
AsnModuleLink	17	0,5%	UnMTF	17,47	10,8%
AddXMLname	11	0,3%			

Figura 28 - Resultados da execução do teste T9.

- T10: execução do BlastcBlk512, que realiza descompactação em blocos (blocos de 512k)

Função	Page Faults	Percentual	Função	Tempo	Percentual
huffman_decode_memory	2890	76,0%	solicitaBlocoDescomp	53,22	31,1%
NlmReadMFILE	256	6,7%	huffman_decode_memory	30,79	18,0%
readdb_get_sequence	123	3,2%	BlastWordFinder_mh_contig	29,93	17,5%
readdb_get_defline_ex	101	2,7%	UnBWT	19,37	11,3%
memset	65	1,7%	UnMTF	18,13	10,6%

Figura 29 - Resultados da execução do teste T10.

6.3 Conclusão

Neste capítulo foram apresentados os resultados obtidos experimentalmente com as diferentes variações da proposta de implementação detalhada no capítulo 5.

Com os dados coletados foi possível observar que entre a versão original e a versão com gerência de memória sem compactação, o número de operações de E/S permaneceu na mesma ordem de grandeza, mas foi agregada uma sobrecarga no tempo total de processamento. Isto ficou mais evidente no segundo cenário onde foi testado um banco de dados de maior volume. Também vale ressaltar algumas variações podem ter sido causadas por variáveis não

controláveis, como, por exemplo, uma ligeira variação na quantidade de memória disponível em cada teste.

Entre as versões alteradas (i.e. compactadas com descompactação total e com descompactação em bloco) pode ser comprovado que a variação no tamanho do bloco reflete de maneira indiretamente proporcional no numero de operações de E/S, ou seja, quanto maior o tamanho do bloco, menor o número de leituras realizadas. Isto acontece porque a taxa de compressão do arquivo é maior quanto maior o tamanho do bloco, e também porque há uma redução no número de chamada às funções de descompactação.

Porém, em todos os cenários observamos que o tempo de execução das versões do BLASTP modificadas foi maior do que o tempo de execução do BLASTP original. Esse resultado é justificado pelo custo associado à operação de descompactação do algoritmo BWT. Este custo representa duas vezes o custo da total execução do algoritmo original do BLASTP. Em resumo, o custo da descompactação pode ser visto como um custo de execução de um novo programa BLASTP. Desta forma, se faz necessária a busca de uma solução para redução do tempo de execução da descompactação usando algoritmo BTW em memória.

Para poder concluir definitivamente que com o uso de compactação para o programa BlastP com o algoritmo BWT não será possível obter melhor desempenho em quanto ao tempo de execução se faz necessário realizar mais testes variando os parâmetros como:

- Tamanho de bloco de compactação;
- Quantidade de *bytes* a descompactar em cada chamada.

Por outro lado, o uso de outros algoritmos de compactação também apresenta uma possibilidade de obter variação nos tempos de execução obtidos.