RECIIS
Electronic Journal of Communication Information & Innovation in Health
[www.reciis.cict.fiocruz.br]
ISSN 1981-6286

SUPPLEMENT – BIOINFORMATICS AND HEALTH

Original Articles

BioProvider implementation issues: a non-intrusive tool for process scheduling and biological data access

DOI: 10.3395/reciis.v1i2.Sup.99en

Maíra Ferreira de Noronha
Departamento de Informática da Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil
maira@inf.puc-rio.br

Sérgio Lifschitz
Departamento de Informática da Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil
maira sergio@inf.puc-rio.br

Abstract
This work discusses the implementation of BioProvider, a tool that efficiently provides data for biological applications. It uses ad-hoc buffer management policies and specific process scheduling control to deal with very large sequence data. We explain how to consider a non-intrusive approach in order to encourage the use of BioProvider in a transparent way while keeping the original applications unchanged. The first instantiation of BioProvider is tailored to BLAST, the most popular sequence comparison program.

Keywords
Process scheduling, biological data access, BioProvider

Introduction

This work proposes a non-intrusive implementation strategy for a data-oriented tool that provides biological sequences, efficiently manages buffer and allows process scheduling control when executing with biological applications. This tool, called *BioProvider*, is kept transparent for all applications, as the communication with the latter is done through a device driver that replaces all read and write function calls to the sequence database files.

One of the most important tasks for the analysis of molecular biology data is sequence comparison, which is the basis for more elaborated manipulations. Information of gene and protein functionality, the position of genes on chromosomes and other information are inferred through the comparison with known sequences, stored with the corresponding headers in databases such as Genbank [Benson et al., 2005] and SWISS-PROT [SIB, 2007]. The programs of the BLAST family [Altschul et al., 1990] are the preferred and most popular comparison tools in these cases.

This paper details the instantiation of BioProvider for running with BLAST programs. Our tool can be straightforward extended in order to deliver data for other applications, use distinct buffer management policies and provide data in different formats from those read by the applications, converting it during runtime. In particular, we show here a great advantage of the non-intrusive approach: BioProvider can be used with two (or more) different BLAST implementations, NCBI BLAST [NCBI, 2007] and WU-BLAST [WU, 2007], which are the most popular ones.

This article is organized as follows: in the next section, we first discuss a buffer management proposal specific for BLAST, one chosen instantiation well ex-

plained here. Then, Section 3 describes the BioProvider architecture, taking into consideration the most suitable page substitution policy. Section 4 gives further details with respect to process scheduling. Finally, Section 5 shows some experimental results that illustrate the advantages of running BLAST in the presence of BioProvider and Section 6 concludes and lists some possible future work.

Buffer management for BLAST

The BLAST program consists of local alignment heuristics for biosequences and is used for the comparison of query sequences with sequence databases. These databases have specific formats and are composed of three files: a sequences file (*.psq), another file with the corresponding headers (*.phr) and an index file (*.pin) that associates the sequences with the corresponding headers [WU, 2007]. The BLAST database is originated from a text file in the FASTA format [NCBI, 2007], using the *formatdb* tool that is provided with BLAST.

The BLAST basic strategy has three stages and is described in Altschul et al. [1990]. During the second stage of the algorithm, the sequence file is fully scanned. It is also the stage where most disk accesses are made. As BLAST reads directly from the operating system files, the sequence files are read in an inefficient way in some common situations. Indeed, this happens when the sequence file does not fit entirely in main memory. Many processes are running at the same time, as no specific buffer sharing techniques are used. Therefore, parts of the sequence database will be copied from disk to memory multiple times during the second stage for the execution of each process.

This second stage of the algorithm has also the following characteristics:

• The sequences (and corresponding pages) in the database are sequentially read, from the beginning to the end. Therefore, it is possible to guess the next pages that will be read and copy them to the buffer beforehand (prefetching).

• The order in which the query sequence is compared with each sequence in the database does not matter. A BLAST process can begin the comparison at any sequence of the database as long as, by the end of the stage, every sequence has been compared.

By taking the described characteristics into account, (LEMOS et al., 2003) suggested an *ad-hoc* buffer management strategy for BLAST. The idea involves the use of memory structures for sequence storage named rings. These consist of memory buffers to which the database sequences are copied. Updates follow a FIFO-like page substitution policy. While present in this ring, data is shared by all the running processes that are on the second stage. Ring pages may be refreshed when all processes have already read the corresponding information.

One clear advantage of this strategy is to allow each BLAST process, once activated, to start reading the sequence file from the first data page available at the ring – not necessarily the first in the sequence file – as the order in which the sequences are read does not matter. The beginning of the sequence file is copied again to the ring as soon as each reading cycle of the file is finished. Therefore, the new process will be able to read all the sequences before the actual first sequence read, regarding completeness.

This strategy is very interesting for use with BLAST because it allows not only data prefetching but also buffer sharing, reducing the read time of sequence data from disk to memory. BioProvider is instantiated in this paper using a similar strategy, as will be described in the following section.

The non-intrusive approach for bioProvider

In order to provide data to molecular biology programs, the BioProvider tool can be implemented in two different ways, whether changing (intrusive) or keeping (non intrusive) the original program source code. The implementation of the intrusive method requires the substitution of each read function call on the code for other functions that communicate with a data provider process. The latter will, on his turn, manage the buffer in memory. In MAURO et al. (2005), the usage of a device driver was suggested for the implementation of the non-intrusive method, so as to simulate the database files and carry out the communication between the BLAST and provider processes.

A device driver is a software layer of the operating system that allows the communication between applications and hardware and software devices, hiding the way the direct communication with the devices is done. The idea proposed in MAURO et al. (2005) is to substitute the biological database files for special files (called device files) associated with a character device driver. By executing the open and read functions for these files, the molecular biology applications execute instead the functions implemented by the driver, that carry out the communication with the database provider process.

In this work, we have chosen to implement the buffer management in a non-intrusive way through the creation of a device driver. By this means, it is not necessary to modify the application source codes. Moreover, the tool can be used with different BLAST versions without needing to change its own source code. We have also chosen to implement the Linux device driver as a kernel module. BioProvider works with BLAST processes that read amino acid databases.

In order to use BioProvider, the BLAST database files must be substituted by device files associated with the device driver. When executing the open and read functions for these files, the BLAST processes execute instead the functions implemented by the driver that carry out the communication with the database provider process and control concurrency and process blocking. Figure 1 shows the architecture of BioProvider and the communication with BLAST:

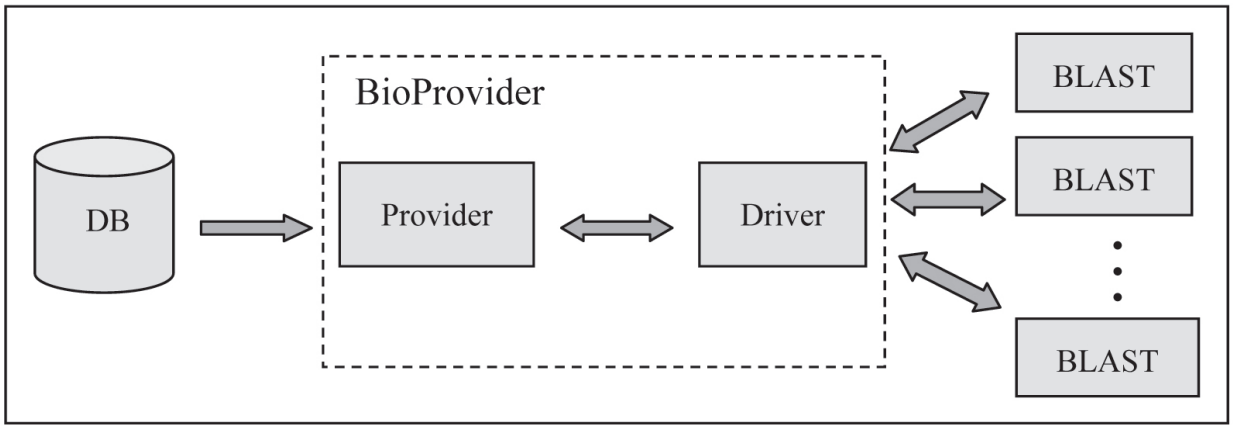

Figure 1 - The BioProvider architecture.

Concerning the BioProvider architecture, a database provider process manages the ring in main memory and must answer all read requests from BLAST processes. The communication between the latter and the provider process is done through functions implemented in the driver. The original database files for BLAST are replaced by special files associated with the driver. Therefore, once these files are accessed as they were the original database, the BLAST processes run actually the driver's functions. The provider process knows the exact location for files and eventually their contents become available. These processes wait in blocked state for the requested information. The provider process wakes up when the information is available. The provider process plays the role of BLAST processes scheduler, deciding which process is executed first.

Buffer management here is similar to the strategy presented in Section 2. Only one main memory ring is considered, multiple rings still need to be evaluated for future versions. The sequences file contains all sequences, one after the other, in compressed format. These are fetched into the ring incrementally. The BLAST processes that are still in the sequential reading step start to read from the content already present in the ring. Thus, if the beginning of the input file (that corresponds to the very first sequence) is not at the ring when the process requests it, the provider replies with the start position of some sequence in the ring.

It is mandatory that BLAST processes start accessing the sequence files at the beginning of a given sequence. Otherwise, they would possibly interpret parts of sequences as full sequences. To avoid this situation, the provider process must identify the beginning of sequences in the file. This may be done by identifying within the sequences file where sequences start. This point is the same for those 2 versions previously studied. Once the sequences file reading starts, all read-requests then will get as the answer the content of a misplaced file position. The requested position is added to the position where the process has started reading. When the content from the end of file is obtained, the next content would be from the beginning of the file. This way the process would read also the content where reading had started. When all active processes would have gone through a given ring position, this position is updated from the sequences file.

An important characteristic present in this buffer management policy is that the information that will be fetched is copied to buffer before the actual request. The buffer update follows a FIFO – first in, first out – strategy. However, buffer regions with random sizes may be updated all together, instead of updates with fixed size.

Another relevant observation is related to the buffer management strategy. Processes may get information that is different from the one originally requested, what is not the case for typical buffer management techniques. Processes receive "fake" file positions in order to focus on a non-intrusive BLAST version. All processes that have started to read the sequence files from some position that is not the first one will have a slightly different view from the actual file. This would lead to execution problems if the provider process and the driver control only read from the sequences file. Other database files are provided by the operational systems, with no modification. Indeed, as the index file has links to the sequences file, these would be pointing at wrong positions of the file that BLAST uses. The index file links sequences in the sequences file to these file annotations in annotation-files. We can check in Figure 2 that this information will create wrong file associations.

A possible solution to this problem would be to add control when the index file is read. Its content could then be modified on-the-fly before sending it to each BLAST process. This way the pointers would not be misplaced. This alternative is feasible but brings some possible drawbacks, besides a higher cost for CPU and memory use. However, the worst part would be that this way one must know the actual index file format in order to modify it. This format varies among distinct implementations and BLAST versions more than the sequence file

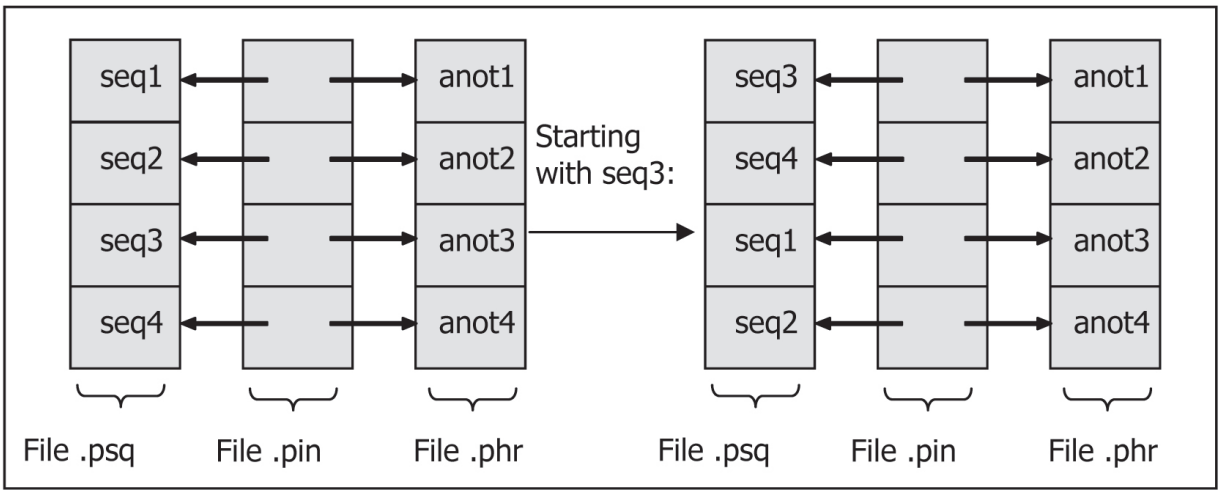

Figure 2 - Misplaced pointers with formatted BLAST files.

format. Therefore, the database server program would need to adapt and change considerably to work with all BLAST flavors.

We have decided here to adopt another strategy: we will pre-process the database in FASTA format and create multiple "images" of the sequence file, each copy corresponding to a different order for reading these sequences. To do this we need only to modify the sequence order in the FASTA file just before creating distinct database instances with BLAST usual tools. We keep all file instances, except the sequence file, which needs only one instance. To avoid the creation of a large number of instances, it is important to give limits to the sequence file positions, where processes start reading. For example, we may divide all sequences in *n* blocks with *m* sequences each. BLAST processes will be allowed to start reading the sequences file only at each block start. BioProvider controls also all access to other files (pin and phr). BLAST gets distinct instances for those index and header files, depending whether the reading process has started.

We illustrate this database pre-formatting step in Figure 3. If we choose to divide into 4 blocks, BLAST processes read the sequence file in 4 different block orders: 1-2-3-4, 2-3-4-1, 3-4-1-2 e 4-1-2-3. This corresponds to rotations for these blocks, while the order of sequences remains the same. Each order has its own index and annotation files and these are provided to the BLAST processes.

This approach increases the amount of disk space needed but makes our provider independent of índex and annotations formats. Actually, as the index file is much smaller than other files, and the headers file is rarely accessed – only to obtain annotations for the most similar sequences to the input sequence –, the I/O cost remains about the same even with multiple instances for those files.

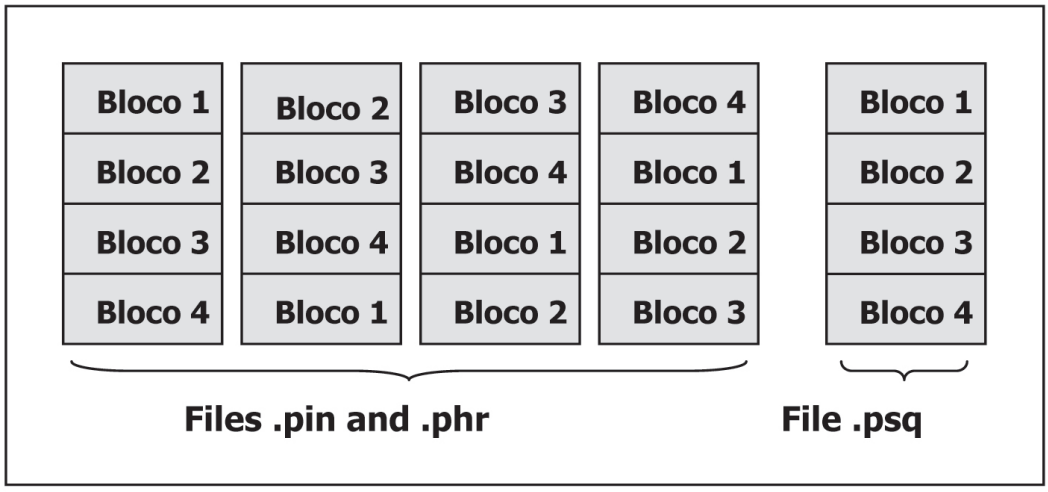

Figure 3 - Database pre-formatting.

When the provider process starts, it executes the read function implemented by the driver and waits blocked for data requisitions. When a BLAST process tries to read the database files, it executes the read function of the device driver, which informs the requested file pages. The read function then blocks the BLAST process and wakes up the provider process, which receives the data requisitions. The provider process manages the ring buffer in memory and provides the requested data by executing the write function of the driver. The data can be read from the ring or directly from the database files. The write function then wakes up the blocked process that requested the provided information. The BLAST process receives the information and continues its execution. Finally, the provider process executes again the read function of the driver and is blocked in order to wait for new data requisitions.

If the buffer management method described in Section 2 is used and the BLAST processes start reading the ring when the present content is not the beginning of the sequence file, the processes will ask for one content of the sequence file and receive another one. In this case, some problems may arise. The problems may happen because the index file has pointers to positions in the sequence file and the headers file that are used to associate each sequence in one file to the corresponding header in the other. The pointers to the sequence file will probably point to a content of the file that is not the original one and the associations with the headers will therefore be wrong.

The best solution found to the described problem requires the limitation of the sequence file positions at which the processes can start reading, by dividing the sequences into n blocks of m sequences. The *formatdb* BLAST tool can then be used to create instances of the index and header files that correspond to each of the orders in which the sequences can be read. As there are only m positions of the sequence file where the processes can start reading, which are the beginning of each sequence block, there will be m instances of the index and header files. The provider process will provide to each BLAST process the files corresponding to the sequence block at which its reading began.

Although this solution augments the disk storage space that is used, it makes BioProvider independent from the index and header file formats, as only the sequence file format must be known by the provider process in order to identify the beginning positions of the sequences. The sequence file format varies very little between different BLAST versions.

Process scheduling

Besides managing the ring in memory, the provider process chooses at each moment the process that will receive the answers. We have created some heuristics as to give preference to processes that are falling behind when reading the ring or requesting data from other database files. An important objective is to assure that all processes will be able to finish their executions, avoiding starvation, which may happen if new processes start all the time, forbidding the refreshment of the ring. This problem can be solved by delaying, from time to time, the response to newly started processes. As these processes have not begun to read the database, they do not hinder the refreshment of the ring, and the other processes can therefore proceed in reading and updating the ring data.

Due to some characteristics of BioProvider and BLAST, some factors have great influence in process performance. The most important factors are database size and the number of BLAST processes running concurrently. If the database fits entirely in memory, the operating system will probably read it from disk only once and maintain the database in memory while processes are reading. This is the most desired situation, in which processes will have better performances. Moreover, in this situation, a buffer management tool will not be needed. On the other hand, if the database is too big and cannot be kept entirely in memory, BLAST performance will deteriorate, as parts of the database will be copied many times to memory when different processes access the same database.

BioProvider improves the performance of BLAST processes by providing them with data that is already in memory and being read by other processes. Therefore, the number of page faults is smaller. The bigger the database size and the number of BLAST processes reading from the same database, the greater are the advantages of using BioProvider, because the probability of processes requesting data that are not available in memory is higher in those cases.

Other key factors that influence process performance are the size of the buffer ring and the number of blocks in which the sequence file is divided. The ring should be big enough to store part of the sequence file data that has not yet been requested and to have, at any given time, the beginning of a block of the sequence file. Thus, ring updates need not be frequent and new processes do not have to wait too long to start reading from the ring. However, if the ring is too big, it may use too much memory and therefore deteriorate the performance of other processes, especially if virtual memory or swapping is needed.

The ideal size of sequence file blocks depends on the size of the buffer ring. If it is bigger than the ring, chances are that several new BLAST processes will have to wait for a block beginning to appear in the ring before they start reading. On the other hand, if the blocks are too small, there will be many instances of index and header files, which may take up much disk space.

The strategies adopted by the provider process to answer read requests and schedule processes have great influence in process performance. Therefore, several strategies were implemented varying the rules for process priorities. Each time the provider process receives a read request, it chooses the BLAST process to answer following certain criteria that are shown below:

1. If the priority is always given to processes that are reading from the buffer ring, one would expect the total

executing time to be smaller, as reading from the ring is faster than reading from disc. However, this strategy does not lead to good results. As each process reads data out of the ring both before and after the ring stage, if there are processes reading from the ring, several processes may be delayed before entering the ring stage or after finishing it. Thus, the implemented strategies give priority to read requests of data out of the ring or alternate giving priority sometimes to processes in the ring stage.

2. After deciding whether to give priority to processes in the ring stage or in another stage, the chosen process is the one that has requested the smallest file position or the data in the smallest ring position. By this means, the processes that have fallen behind in the ring stage can advance and stop blocking ring update.

3. It is important to ensure that the running BLAST processes do not suffer starvation, waiting eternally blocked for the answers to read requests. However, when the provider process starts answering the requests of a new BLAST process, the ring update is blocked for a certain amount of time. This happens because the sequence file block by which the process will start reading is chosen at the moment that it receives the answer to the first read request, as it will read the index file that corresponds to the chosen block before entering the ring stage. If new BLAST processes are very frequent, ring update may never occur. To solve this problem, from time to time the provider process refuses to answer the requests of new BLAST processes, so that ring update can be ensured.

Finally, it is important to mention how input sequences influence BLAST process performance. In average, BLAST processes that have bigger input sequences tend to have longer running time, as more similarities with the database are found. Input sequences that are very similar to those of the database produce the same effect. As some BLAST processes are faster than others, at any given moment some processes may be ahead in the ring stage, waiting for ring update, while others may have fallen behind and be blocking the update.

This problem can be minimized if multiple rings exist and processes with similar velocities read from the same ring. However, the solution has not yet been implemented. As an alternative, if the input sequences are known beforehand (one input per BLAST process), the BLAST processes can be run ordered by the size of the input sequence. Another solution is to run the processes in separate groups with input sequences of similar size.

Preliminary results

We have implemented BioProvider initially for Linux core 2.4 and 2.6, such as Linux Fedora 3 and 4, and Linux Red Hat 8. BioProvider main modules are a data provider program, a data driver and a program that completes the provider's execution. It is worth mentioning that BioProvider may be used either with NCBI BLAST (from version 2.0) and WU-BLAST 1.4, due to the similar sequences file format. We have developed also some tools that automatically prepare the database and that create configuration files. The user may customize the way he uses BioProvider by defining the ring's size in main memory and the number of sequences per block.

Preliminary practical results were obtained in order to evaluate NCBI-BLAST in the presence of BioProvider. We have run our experiments on a 3 GHz Pentium 4-based computer with 512MB of RAM. The sequence database initially considered was *nr*, the protein database available at [NCBI, 2007]. This is one of the most important databases for biologists, using over 1.2GB and it contains information about many different organisms. Its protein sequence database size, for these experiments, was about 1.3GB.

We have evaluated the runtime for 50 BLAST processes varying the amount of available RAM and the ring size, each BLAST process starting about 1 minute one after the other. We have divided the *nr* database in 5 blocks, each containing about the same number of sequences. With the help of GRUB (boot loader) program, we could configure the RAM actual size. Our 50 input sequences were randomly chosen from *ecoli.aa*, *swissprot* and *ptaa* databases, and each BLAST process used one of these as its query sequence. Next, we present some of our preliminary test results:

Table 1 – BLAST normal execution

RAM size	256M	512M
Average Runtime (secs)	2:40:57	1:11:15

Table 2 – BLAST execution with BioProvider

RAM size	256M			512M		
Ring size	25M	50M	100M	25M	50M	100M
Average Runtime (secs)	27:55	29:24	42:11	24:05	34:45	25:26
Speedup	83%	82%	74%	66%	51%	64%

From tables 1 and 2 we can observe that those 50 BLAST processes running in the presence of BioProvider have obtained from 51% up to 83% of improvement. Many other results, also very positive for BioProvider, may be checked in NORONHA (2006).

Conclusions

An instantiation of BioProvider was implemented to efficiently provide data to BLAST by doing buffer management, data controlling and process scheduling, taking into account specific characteristics of BLAST database access. BioProvider was implemented with a non-intrusive approach through the usage of a device driver that carries out the communication between processes. Therefore, the application source codes can remain unchanged and the tool works at the same time with versions of NCBI BLAST and WU-BLAST. Moreover, BioProvider can be easily extended to provide in the future other database solutions for molecular biology applications.

The tests done with BioProvider showed many situations where it was possible to improve BLAST performance. It was also possible to verify the influence of some factors on BLAST using BioProvider. In future works, the tool can be extended in many points, some of which are listed below:

• To implement the possibility to provide BLAST with nucleotides database files. Similar buffer management and process scheduling techniques can be used.

• To introduce and analyze the performance of other buffer management techniques and strategies of choosing the process to answer. One possibility is the creation of multiple rings in memory, to which processes with similar speed may be allocated, in order to avoid slow processes to hinder the execution of the quicker processes while reading from the ring.

• To use other techniques of providing data to BLAST, such as the dynamic creation of index files to be supplied at each process. This would make unnecessary the division of the database into blocks and the creation of different instances of the index and note files.

• To use BioProvider to provide other database solutions to BLAST. A work to be done is the inclusion of file compression and decompression techniques, in order to enable the storage of data in different formats than those seen by the processes, translating them during runtime in a similar way as ROSA et al. (2007). By this means, to make BLAST more efficient, techniques that include both time management and buffer compression can be developed.

• Extend BioProvider to provide database solutions for other Bioinformatics tools. One of the extensions consists of providing buffer management techniques to the FASTA tool (PEARSON, 1991). As it is also a biosequence comparison tool, FASTA shares many characteristics with BLAST and can benefit from similar buffer management techniques. Other tools can benefit from using BioProvider for different ends, due to its transparency characteristics.

The source code for BioProvider as well as usage instructions can be found on www.inf.puc-rio/~blast.

Bibliographic references

ALTSCHUL, S., et al. Basic Local Alignment Search Tool, **Journal of Molecular Biology**, 215, p.403-410, 1990.

BENSON, D.A. et al. GenBank, **Nucleic Acids Research**, Jan 1;33 (Database issue):D34-8, 2005.

LEMOS, M.; LIFSCHITZ, S. A Study of a Multi-Ring Buffer Management for BLAST, 1st International Workshop on Biological Data Management, In conjunction with DEXA, 2003, p.5-9.

MAURO, R.; LIFSCHITZ, S. An I/O Device Driver for Bioinformatics Tools: the case for BLAST, **Genetics and Molecular Research** (GMR), v.4, n.3, p.563-570, 2005.

NCBI. "NCBI BLAST". Available at: <http://www.ncbi.nlm.nih.gov/BLAST/>. Accessed: 2007.

NORONHA, M.F. "Controle da Execução e Disponibilização de Dados para Aplicativos sobre Seqüências Biológicas: o Caso BLAST", 2006. Dissertação (Mestrado). Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Riode Janeiro.

PEARSON, W. Searching Protein Sequence Libraries: Comparison of the Sensitivity and Selectivity of the Smith-Waterman and FASTA algorithms, **Genomics**, v.11, p.635-650, 1999.

ROSA, J.; LIFSCHITZ, S. "Um Estudo de Compactação de Dados para Biosseqüências", 2006. Dissertação (Mestrado). Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

SIB. "Swiss-Prot and TrEMBL". Available at: <http://bo.expasy.org/sprot/>. Accessed: 2007.

WU. "WU-BLAST". Available at: <http://blast.wustl.edu>. Accessed: 2007.

About the authors

Sérgio Lifschitz

He has obtained an electrical engineering degree (1986), a MSc degree also in electrical engeneering (1990), both from at the Pontifícia Universidade Católica do Rio de Janeiro and Ph.D in Informatics - Databases, from the École Nationale Supérieure des Télécommunications, ENST Paris, France (1994). During 2005 he worked at UC San Diego (USA) as a post-doctoral visiting scholar. He is currently an Associate Professor at PUC-Rio in the Informatics Department. His main research interests are in the field of databases, either with (1) autonomous computation and self-tuning systems or (2) tools and data management systems for bioinformatics applications.

Maíra Ferreira de Noronha

Holds a degree in Computer Engeneering from (2003) and Master in Informatics (2006), both from Pontifícia Universidade Católica do Rio de Janeiro. Her research experience is mainly in the field of databases, bioinformatics and computational intelligence, auto-tuning of databases, genetic algorithms and neural networks. Currently she works as system analyst at Petrobrás.