

CONTRACT NUMBER 508830

DEISA

**DISTRIBUTED EUROPEAN INFRASTRUCTURE FOR
SUPERCOMPUTING APPLICATIONS**

European Community Sixth Framework Programme
RESEARCH INFRASTRUCTURES
Integrated Infrastructure Initiative

JRA4 – Production operation of Genomic Codes

Deliverable ID: D-JRA4-2a

Due date : April 31st, 2005

Actual delivery date: May 15, 2005

Lead contractor for this deliverable: IDRIS – CNRS, France

Project start date : May 1st, 2004

Duration: 5 years

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	X
RE	Restricted to a group specified by the consortium (including the Commission	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Table of Content

Table of Content	2
1. Executive Summary	3
2. Introduction	4
3. Status of global genomics environment and coupling with InfoBioGen.....	5
4. Status of "Identification of new human mitochondrial proteins"	5
5. Status of "Large scale microbial genome re-annotation"	6
6. Annex.....	9

1. Executive Summary

This document is one of the two PM12 deliverables of the Joint Research Activity in Life Sciences. It describes the current production status of the two initial “early users” applications in the Genomics area that are planned in the work program. It also describes the activity being carried to deploy a software bio-informatics environment in the DEISA research infrastructure.

This document is publicly available.

2. Introduction

The area of Life Sciences is one of the most challenging ones in the context of high performance computing. The DEISA strategy aims at enabling and providing support for new, demanding, very high end simulations that go much beyond the traditional bioinformatics applications that are performed today on lightweight platforms, typically Linux clusters. However, in this area the important raw computing power or the data management facilities provided by the DEISA platforms has to be interfaced and integrated with external lightweight elements (Web interfaces, lightweight servers, etc) that are the ones that are accessed directly by the end users.

This is the reason why particular attention has been paid to portals that hide the complexity of the DEISA environment from end users. As explained in D-JRA4-1.a, one of the first steps taken by IDRIS was to deploy a job submission environment that would allow computing centres specialized in Genomics to reroute some of their most demanding job requests to the DEISA platform, in a way transparent to the end users. This activity was not initially planned in the JRA4 work program, but it has been incorporated to it and reported here because of the basic role it plays in adapting the DEISA environment to the needs and requirements of the Life Sciences user communities.

In this deliverable, reference is made to applications, databases and software environments deployed at the IBM supercomputer at IDRIS. Given the tight coupling of the four platforms of the initial "core" distributed supercomputer, it is clear that this whole environment is immediately extensible to the full distributed AIX super-cluster. GPFS will allow, for example, transparent access to the genomic databases that are maintained at IDRIS.

The purpose of this deliverable is to report on the production status of the basic DEISA genomics environment, and of the two "flagship" applications that were proposed in the initial work program.

At this workshop, the JRA4 staff met for a full day with the "DEISA early users" from the genomics community. The intention was to push as much as possible the planned applications to tackle some extreme problems that cannot be handled in other platforms. Scientists required, of course, information on the project status and on resource availability, and came up with new, more aggressive, scientific objectives, that will become "flagship" applications of the DEISA project. This is why a few comments concerning the renewed scientific objectives are included in this deliverable. The rest of this document is organized as follows:

- ? Section 3 describes the present status of the software environment allowing transparent rerouting of BLAST genomics applications from the InfoBioGen Genomics Centre in France to the IDRIS – DEISA infrastructure.
- ? Section 4 describes the present status of the genomics project "Identification of new human mitochondrial proteins".
- ? Section 5 describes the present status of the genomics project "Large scale microbial genome re-annotation".

3. Production status of the portal to the InfoBioGen genomics centre.

3.1 Purpose

The purpose of the activity described here is to deploy a job submission environment that can transfer to the DEISA genomics platform a substantial computational workload in a way totally transparent to end users. Scientists continue to access their traditional genomics computing centre, and are not DEISA aware. A small fraction of applications that can be more efficiently run in the DEISA environment are “rerouted” the DEISA platforms by the system administrators. Results are returned to the genomics centres and presented to the end users in the standard way.

Details on the job migration technologies deployed in this activity, and on the DEISA genomics environment deployed to support these applications – in particular, database installations - were described in a previous deliverable (D-JRA4-1a)

3.2 Status

The portal is completed and ready to start full, intensive, production operation. This is waiting for some strategic decisions related to the issue of resource allocation.

It has to be emphasized that this strategy raises new resource allocation. Genomic computing centres provide fully open services and grant resources to any user that requests them. This works fine because the computational resources engaged are very limited and do not justify detailed accounting and specific allocation policies. DEISA sites, instead, allocate substantial resources to specific projects whose scientific relevance is subject to peer review. In the pilot infrastructure that is being deployed at IDRIS, the solution that is being deployed – approved by CNRS – is to perform “community allocations”, namely, allocating resources to a big community represented by a few responsible scientists, the end user remaining unknown to IDRIS. This is being deployed and tested now, and represents a major departure from the traditional mode of operation of national supercomputing centres.

4. Production status of project “Identification of new human mitochondrial proteins”

Objective

The purpose of this project is the high-throughput identification of new human mitochondrial proteins by “in silico” comparative genomics. The identification of these nuclear mitochondrial genes would allow a better understanding of mitochondrial diseases.

Method

Compare the 208 available bacterial proteomes against all eukaryote genomes of interest such as *Homo sapiens*, *Mus musculus*, *Drosophila melanogaster*, *Caenorhabditis elegans*, (20 complete eukaryote genomes are available at NCBI). The program tblastn, from the well known WU BLAST distribution, will be used for this purpose, together with custom filters to select only the sequences of interest.

Status

Full production status achieved. Large production runs starting in May 2005.

During the last six months, the final optimization of the computational environment for this application has been completed. This project is perfectly well adapted to the DEISA supercomputing infrastructure, in particular to the large IBM shared memory 32 processors computing nodes on the AIX super-cluster. The scientific team is highly motivated, and has worked efficiently with the IDRIS support team.

We are including in the Annex a DEISA Technical Document produced by the IDRIS team that is relatively self-contained and describes all the actions taken to adapt and optimize this application for the DEISA environment.

5. Production status of project “Large scale microbial genome re-annotation”

Objective

Re-annotate all known prokaryotic genomes (194) using the AGMIAL platform developed at the MIG INRA laboratory in France, and provide to the scientific community unified data mining bioinformatics tools to explore this huge amount of data.

Method

Run on the 194 prokaryotic genomes the AGMIAL tool suite and store the annotation results in a relational database. AGMIAL integrates numerous free software (as BLAST), including a sophisticated and time-consuming fold recognition method (FROST).

Starting in 2000, a number of laboratories from the French National institute of Agronomic Research (INRA) embarked in a project called AGMIAL, whose purpose was to sequence and analyze the genome of a number of bacteria relevant for the food-processing industry. AGMIAL is a French acronym for “Annotation de Genomes Microbiens d’Intérêt Agro-alimentaire”, which translates into English in “Annotation of Microbial Genomes of Importance for Farm-produce Industry”

The resulting relational database (the estimated size is a few terabytes) will be located and maintained in the DEISA environment. IDRIS will provide the environment needed to make this data public in a way compatible with the security of the DEISA environment,

using tools and methods already in operation for the climate community. This database will be accessed both by biologists and by bio-informatics research groups.

Status:

There are two issues in this application:

a) The database:

Deploying, maintaining and providing public access to the database, in a context that is compatible with the very strict security policies of the national supercomputing centres. This problem is under control, IDRIS can provide the physical storage support and the software technologies to enable this service.

b) The AGMIAL tool suite:

This set of codes has been installed on the AIX super-cluster in the DEISA platform, and substantial time has been spent in trying to optimize it for the DEISA platform. The AGMIAL tools include not only a number of public domain packages but also a number of control procedures needed to steer a complex chain of treatments.

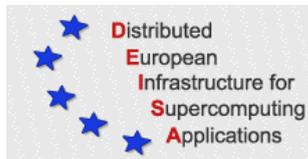
The fundamental problem we have met here is that these codes are absolutely not adapted in their present form to the scope of the computations that are required for this project. The codes have been developed having cheap lightweight platforms in mind, they incorporate an impressive number of Perl scripts that individually do almost nothing, and the whole tool suite is a patchwork of applications and codes not adapted to a supercomputing environment. The AGMIAL suite does not “scale” to a supercomputer in its present form, and using it “as such” in the DEISA environment would be wasteful.

It is of course possible to perform a careful migration of this suite to the DEISA environment and, given the scientific interest of this project, DEISA is prepared to provide the effort needed to do so. However, before embarking human resources in this activity, a fundamental issue concerning the relation with the INRA scientific team has to be clarified. DEISA provides advanced user support for enabling, deploying and optimizing an application, but we cannot provide to scientific users a permanent support service. Given the important effort required by the migration, we have to make sure that INRA will develop in the future the competences to maintain and operate the supercomputing AGMIAL suite. We need to protect our human investments, and we are asking guarantees of a more important involvement on the computer science aspects of this project before we proceed. Discussions with INRA are under way. If we estimate that the guarantees we are asking are not sufficient, the project will be dropped, and replaced by one of the new proposals emerging from the Extreme Computing Initiative.

List of Acronyms and Abbreviations

AGMIAL	Annotation de Génomes Microbiens d'Intérêt Agro-Alimentaire
BLAST	Basic Logical Alignment Research Tool
IRISA	Institut de Recherche en Informatique et Systèmes Aléatoires
INRA	Institut National de la Recherche Agronomique
INSERM	Institut National de la Santé et de la Recherche Médicale
NCBI	National Centre for Biotechnology Information

6. Annex: DEISA Technical Document on the “Mitochondrial Proteins” application



High throughput identification of new human mitochondrial proteins by *in silico* comparative genomics

Status of Project Support, Planning and Advancement Report

DEISA TECHNICAL DOCUMENT
April 2005

Isabelle DUPAYS, Olivier GLORIEUX, Anne FOUILLOUX, Denis RAUX
IDRIS - CNRS

Table of Content

Table of Content	1
1. Introduction	2
2. The NCBI Blast code	3
3. Strategies for parallelisation of similarity search algorithms	7
4. Setup of the BLAST simulation.....	8
5. Annex A: Limitations on the number of concatenated queries.....	12
6. Annex B: BLAST report output	14
7. Annex C: Perl script for FASTA database segmentation	17
8. Annex D: Perl script for BLAST database updates	19

1. Introduction

1.1 The DEISA Life Sciences research activity

One of the purposes of DEISA's Life Sciences Joint Research Activity is to enable leading, ground breaking genomics simulations, by providing scientist with access to the most powerful European supercomputers, as well as advanced user support on migration, optimization, and other computational sciences issues.

The 'High throughput Identification of new human mitochondrial proteins by *in silico* comparative genomics" is one of the projects to which IDRIS, as leader of the Life Sciences research activity, is providing this advanced support. This project is driven by a scientific team from INSERM U694. The purpose of this technical report is to document the work done in enabling and optimizing the simulations related to this project.

Our first objective was to port the NCBI-BLAST application on our scalar supercomputer (a IBM-SP4 platform called Zahir) with 8 P690 SMP nodes with 32 Power4 processors each, 12 P690+ SMP nodes with 32 Power4 processors each, and 96 P655+ nodes of 4 processors each. Our second objective was to optimize this code in order to provide the best computer platform for similarity search applications. It turns out that, for the project under consideration, the 32 processors shared memory nodes are the most efficient ones.

1.2 Scientific objectives of the project

Background:

The human mitochondrial proteome is believed to contain over a thousand proteins. However, the size of the mitochondrial proteome differs markedly among species. Recent studies have demonstrated the dual origin of the mitochondrial proteome in yeast; 50-60 % of the mitochondrial proteins have homologues in prokaryotic species whereas 40-50 % do not [TFM+04]. The current hypothesis is that these proteins may have been recruited from pre-existing nuclear genes and targeted toward mitochondria. The large number of eukaryote-derived genes suggests that numerous specific functions of modern mitochondria were absent in prokaryotic mitochondrial ancestors. Thus, the modern mitochondrial proteome is composed of proteins with a dual eukaryotic and prokaryotic origin, the eubacterial proteins being encoded by two genomes, nuclear and mitochondrial.

Hypothesis:

It was previously found that 88 % of human mitochondrial proteins with prokaryotic homologues, noted PH+, were significantly larger (average:435 aminoacids, range: 69-1500) than their prokaryotic homologues (average: 400 aminoacids, range: 38-1353) ($p < 0.0001$). This difference was mainly due to the presence of a supplementary

N-terminal sequence in most of proteins. These additional sequences displayed no homology to prokaryotic proteins and were probably of eukaryotic origin. Thus, most of the PH+ proteins are in fact probably prokaryotic-eukaryotic chimeral proteins. Indeed, only half of human mitochondrial proteins are known at present and the identification of 500 additional proteins will be a capital task in the next few years.

Method:

The project needs to compare the 208 available bacterial proteoms against all eukaryote genomes of interest (such as Homo sapiens, Mus musculus, Drosophila melanogaster, Caenorhabditis elegance-20 complete eukaryote genomes are available at NCBI). The program tblastn, from the well known NCBI BLAST distribution, will be used for this purpose, together with a custom filter to only select the sequences of interest. The computational time, using default tblastn parameters, is estimated to 420 days on a local SunFire 280R, dual proprocessors/4Gb RAM, for the human genome and 98 bacterial proteoms only; Using more accurate tblastn parameters, would require greater processing capabilities. In addition, the genomic data has considerably increased and it would be interesting to perform our analysis on a new and uptaded genomes. Consequently, if a phylogenetic study is required, multiple alignements software will be used.

2. The NCBI-Blast code

2.1 General comments

BLAST is an heuristic search algorithm employed by a number of genetic search tools. These tools are used by researchers to identify similarity between genetic sequences. Typically, there is an input sequence that is BLASTed against a database of known sequences - the target set. The result of a BLAST search is a list of sequences from the target set that was found to have significant matching regions of the input sequence. In large-scale sequencing projects, this data can be useful to determine if a particular sequence has already been discovered or if contamination has occurred. BLAST can also be useful in a broader sense by providing insight into a sequence's function by matching it with some sequence of known function.

The specific implementation of BLAST used at IDRIS is NCBI BLAST (freely available from <ftp://ncbi.nlm.nih.gov/blast>). The NCBI BLAST code is written in C and is multi-threaded.

NCBI BLAST (Basic Local Alignment Search Tool) is a set of similarity search programs designed to explore all of the available sequence databases regardless of whether the query is protein or DNA. The BLAST programs have been designed for speed, with a minimal sacrifice of sensitivity to distant sequence relationships. The scores assigned in a BLAST search have a well-defined statistical interpretation, making real matches easier to distinguish from random background hits. BLAST uses a heuristic algorithm that seeks local as opposed to global alignments and is therefore

able to detect relationships among sequences that share only isolated regions of similarity.

NCBI BLAST recognizes two types of sequences: nucleotide and peptide. Both sequence types are represented as a string of ASCII characters. Nucleotide sequences are made up of four letters (A, T, C, and G). These represent the 4 bases in DNA - Adenine, Thymine, Guanine and Cytosine. Thus, a short nucleotide sequence input into BLAST might be ACCTGTATGTGA. This string also codes for the complementary DNA strand TGGACATACACT (A bonds with T, C bonds with G). One can imagine these two sequences of nucleotides being placed in parallel and then twisted to create the familiar DNA double helix. For nucleotide to nucleotide queries NCBI BLAST takes the complementary strand of the query into account when searching. A peptide sequence (a protein sequence) is also made up of a string of letters but, in this case, each represents one of the twenty amino acids. Peptides are encoded by triplets of nucleotides, as specified by the genetic code. There are three possible reading frames (+0, +1, +2) in both directions, and therefore there are six ways to translate a nucleotide sequence into a peptide sequence.

2.2 - Selection of the BLAST program

The NCBI BLAST distribution contains five variations of BLAST:

- ? blastn
- ? blastx
- ? tblastn
- ? tblastx
- ? blastp+

The appropriate selection of a BLAST program for a given search is influenced by the following three factors:

- ? the nature of the query,
- ? the purpose of the search,
- ? the database intended as the target of the search.

The code **blastn** compares a nucleotide sequence against a nucleotide database and is relatively quick. The code **blastx** compares a nucleotide sequence against a protein database. To do this, the nucleotide subject needs to be translated into a peptide sequence. Since there are six different translations, the basic BLAST algorithm must be applied six times to complete the query. Like **blastn**, **blastx** compares a nucleotide sequence to a nucleotide database but in this case each is translated (in all 6 reading frames) into a peptide sequence before BLAST search. This is the most computationally intensive of the blast programs since the BLAST algorithm must be invoked 36 times for each sequence to sequence comparison. The program **blastp** compares a peptide sequence to a peptide database and is relatively quick. The program **blastn** compares a peptide sequence against a nucleotide

database. As with **blastx**, each sequence to sequence comparison requires six calls to BLAST.

2.3 - The NCBI BLAST algorithm

In order to identify what could affect the runtime of the parallel SMP NCBI BLAST code (uses a multithreading approach), we need to understand the parallel BLAST algorithm:

A high-level sketch of the parallel algorithm is given below:

```
BEGIN
// stage-1:  compile a list of highscoring words
Parse command line arguments;
Build word list and DFA (Deterministic Finite-state Automaton);

// stage-2:  search on database using word list
Create a new task;
Fork threads to run on different processors {
    Sync processors;
    Init task into  chunks;
    Allocate chunks on demand to different processors;
    Each processor do search and extend on the
    sequences in the task chunks allocated;
}

// stage-3:  collate sort and print hits
Collate hits
Sort hits
Print search reports
END
```

The BLAST algorithm is done in three stages:

- ? compile a list of high scoring words
- ? scan the database for hits by matching against the word list and extend all the hits found to a longer match
- ? collate sort and print the hits obtained

In the first stage the BLAST algorithm compiles a list of words based on the input sequence. Each word is a short segment of the input sequence that are likely to have some statistical significance. This stage is done serially. In the second stage the parallel BLAST algorithm scans the database for hits by dividing the database into

500 task chunks and allocating these task chunks on demand to the processors. Each processor looks for hits by matching the word list against the sequences in its task chunk and tries to extend each of these hits to longer matches. When a processor finishes its chunk, the master processor allocates another task chunk to it. In the third stage the master processor collates, sorts and outputs the hits.

The possible factors that could affect the run time are the sequence itself, the amount of resource contention in the second stage and the amount of serial component in the algorithm.

2.4 - Important NCBI Blast parameters

a) Filter parameters:

To select alignments with E-Value of 0.0001, we used the **-e** parameter of the **blastall** program:

```
-e Expectation value (E) [Real]
  default = 10.0}
```

b) Number of concatenated queries

A new option has been added to search multiple queries at once for the **blastn** and **tblastn** program options of **blastall**:

```
-B Number of concatenated queries, for blastn and tblastn [Integer]
  Optional
  default = 0
```

This new feature is similar in principle, but different in implementation from the support for multiple queries already existing in megablast. The combination of ungapped search (**-g F**) and multiple queries (**-B N**) is not supported. The argument to **-B** option must be equal to the number of sequences in the FASTA input file.

Processing multiple query sequences in one run can be much faster than processing them with separate runs because the database is scanned only one time for the entire set of queries. When the **-B** option is used, the results may differ from the ones produced with individual queries. Usually results will be at least as good or better (in terms of score/evaluate) than the results of the corresponding individual queries; exceptions occur due to the heuristic nature of BLAST. Additional alignments may appear. It is guaranteed that matching sequences will appear in the same order when they are tied in evaluate and are part of the output both with and without **-B**. When the **-B** option is used, the summary statistics at the bottom of the output are for the combined set of queries; at present, the summary statistics are not tabulated for the individual queries in a multiple-query input.

2.5 – Enhancements

a) Allowing a larger number of concatenated queries

Actually, there is a strong limitation on the number of concatenated queries with the NCBI Blast version 2.2.9 (this limitation still exists with the version 2.2.10). The number of concatenated queries is limited to 255 because all the corresponding variables are declared as **unsigned char**. This value is sufficient for small platforms where the amount of available memory is small. On our IBM-4, much larger memory can be used and we modified the BLAST code in order to allow concatenation of more than 255 queries in **[t]blastn** programs.

All changes are detailed in Annex A.

b) Changing BLAST report Output

We are using the standard BLAST report output (option `\verb+-m 0+`) and we only change the report output for this particular option.

The algorithm of the standard BLAST report output is given below:

```
BEGIN
// Stage-1
Prints Header
// Stage-2
Prints On-line summaries
// Stage-3
Prints Alignments
// Stage-4
Prints Footer
END
```

When using the **-B** option (concatenated queries) all the query names are listed, and then all the on-line summaries are given, followed by the alignments. Finally, one footer is produced for the whole report. This is then difficult to discern which alignments belong to which query. As using the **-B** option is a key enhancements in terms of CPUs, we decided to change the BLAST report output. The new algorithm is then:

```
BEGIN
// Stage-1
Prints Header
Loop over queries {
  // Stage 2
  Prints On-line summaries
  // Stage-3
  Prints Alignments
}
// Stage-4
Prints Footer
END
```

This minor change allows us to exploit the **-B** option. Changes are detailed in Annex B.

As a part of the collaboration between the DEISA and EGEE projects, joint applications are being deployed in the area of computationally intensive bio-medical applications.

3. Strategies for the parallelisation of similarity search algorithms

Sequence comparison algorithms and tools, and especially BLAST (Basic Local Alignment Search Tool), is one of the corner stones of bioinformatics. However, sequence databases are exploding in size, growing at an exponential rate (currently doubling in about 14 months i.e. exceeding Moore's Law for microprocessors which is about 18 months). Therefore, the parallelization of BLAST is crucial to improve the effectiveness of sequence comparison.

There are two methods by which BLAST can be parallelized to achieve linear or even super-linear speedup:

a) Database Segmentation:

In this model, a large sized database is split as several nearly equal sized databases. Each node stores one part of the database. The independent segments of the database are searched on each processor or node, and results are collated into a single output file. Several implementations of database segmentation exist, the first of which was within NCBI's BLAST itself. NCBI-BLAST implements database segmentation by multi-threading the search, each processor in a SMP node being assigned a distinct portion of the database.

b) Query Segmentation

Query segmentation splits up a set of query sequences such that each node in a cluster or CPU on an SMP system searches a fraction of the query sequences. By doing so, several BLAST searches can execute in parallel on different queries. BLAST searches using query segmentation on a cluster typically replicate the entire database on each node's local storage system.

When the database fits in core memory, however, query segmentation can achieve near linear scalability for all BLAST search types, even on SMP architectures.

4. Setup of the BLAST simulations

4.1 - Database Management

Biogenetic software access databases for information . This rises several kinds of problems

- ? The database access time may be quit slow compared to the execution time.
- ? Data must be up to date.
- ? Data be shared between computation nodes for parallel runs.
- ? Storage requests evolve progressively as new applications come

All of this leads IDRIS to decide to store the databases permanently on a shared file system within the computation nodes (GPFS technology).

The first database which has been fetched is the blast-index. This was a simple mirroring of a directory of INFOBIOGEN (\verb+www.infobiogen.fr+). The difficulty was the amount of data and the daily refresh. We have developed a perl script which use the public software wget to bring back files. The goal of the script is to ensure the integrity of the copy regarding sizes and dates after having transferred all the files. In fact, it appears that files was modified just after the transfert. This put the image wrong, especially at the end of the months when almost the whole database is updated. The script keep transferring until the copy is perfect. The volume of this base is 43Go.

For the remain of bases the work had been more complex, because a processing must have been made on the files in order to get them in the right format. This suppose a particular development to ensure the synchronisation of the data which change too.

To summarize, if the data fetching will increase, the script must be redesigned to be more reusable and performant. A multitasking version should be made to process data during the transfert phases. And we will have to think on what data we would keep and what we wouldn't.

The perl script for Blast databases update is described in appendix D.

4.2 - Input data

The objective is to compare peptide sequences against nucleotide sequences:

Peptide sequences: allbact.faa contains 644579 protein sequences which correspond to the 208 available bacterial proteoms; Nucleotide sequences: various eukaryote genomes, in particular the *Homo sapiens* genome (hg17, 24 chromosomes), *Mus musculus* (mm5, 21 chromosomes), *Drosophila melanogaster* (dm2, 5 chromosomes) and *Caenorhabditis elegans* (ce2, 6 chromosomes).

For **tblastn**, the basic BLAST algorithm must be applied six times to complete the query because the nucleotide sequence needs to be translated into a peptide sequence (there are six different translations).

4.3 - Sensitivity studies

a) Database segmentation

Our objective was to find the optimal length of the subject sequences for the **tblastn** program i.e. for which the CPU time is the smallest.

We needed a script to split the original subject sequence into smaller sequences, keeping the original format of the data (FASTA). Also, in order to be sure not to miss hits, the script had to be able to create an overlap of a given size between two created sequences. There are several scripts proposing those classic features but none proved to be basic and low-level enough to be efficient in our configuration, this is why we made one : `splitSeqIdris.pl` sing Perl.

To test the best subject length we chose the entire human genome with which we would BLAST a set of 256 bacterial proteins.

First we split with the in-house script the genome, with the ten following splitting values: 10000, 50000, 75000, 100000, 125000, 500000, 750000, 1000000, 5000000, 10000000.

```
cat hg17 | perl splitSeqIdris.pl -tm 10000 > hg17_10000
```

The Perl script for Fasta databases segmentation is described in appendix C.

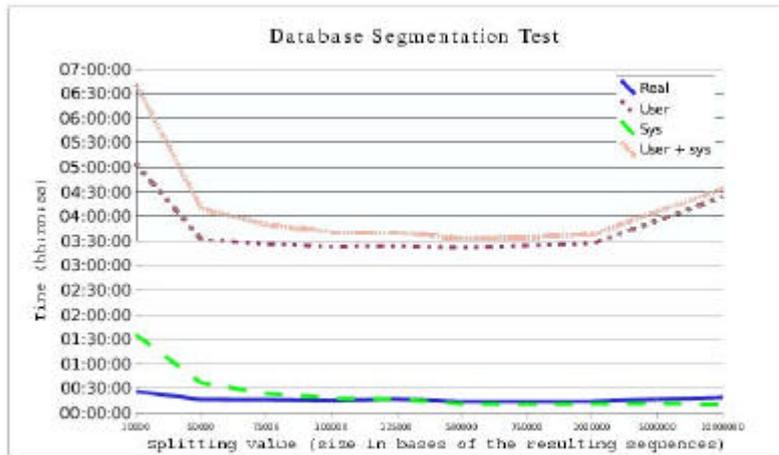
Then we formatted the new FASTA databases with `formatdb` (BLAST's formatting tool)

```
formatdb -i hg17_10000 -p F
```

and finally, the ten BLASTS

```
blastall -p tblastn -d hg17_10000 -i NC_254.faa -a 16 -e 1e-4 -o  
mitopidHg17_1.B.out -B 254
```

Feuille1



Page1

The best segmentation value appears on the graphic to be around 500 000 bases and since we have no particular constraints it is the value that we will use. In this case, the elapsed time was about 13 minutes, and the total time (cpu + sys time) was about 3h30.

b) Number of concatenated queries

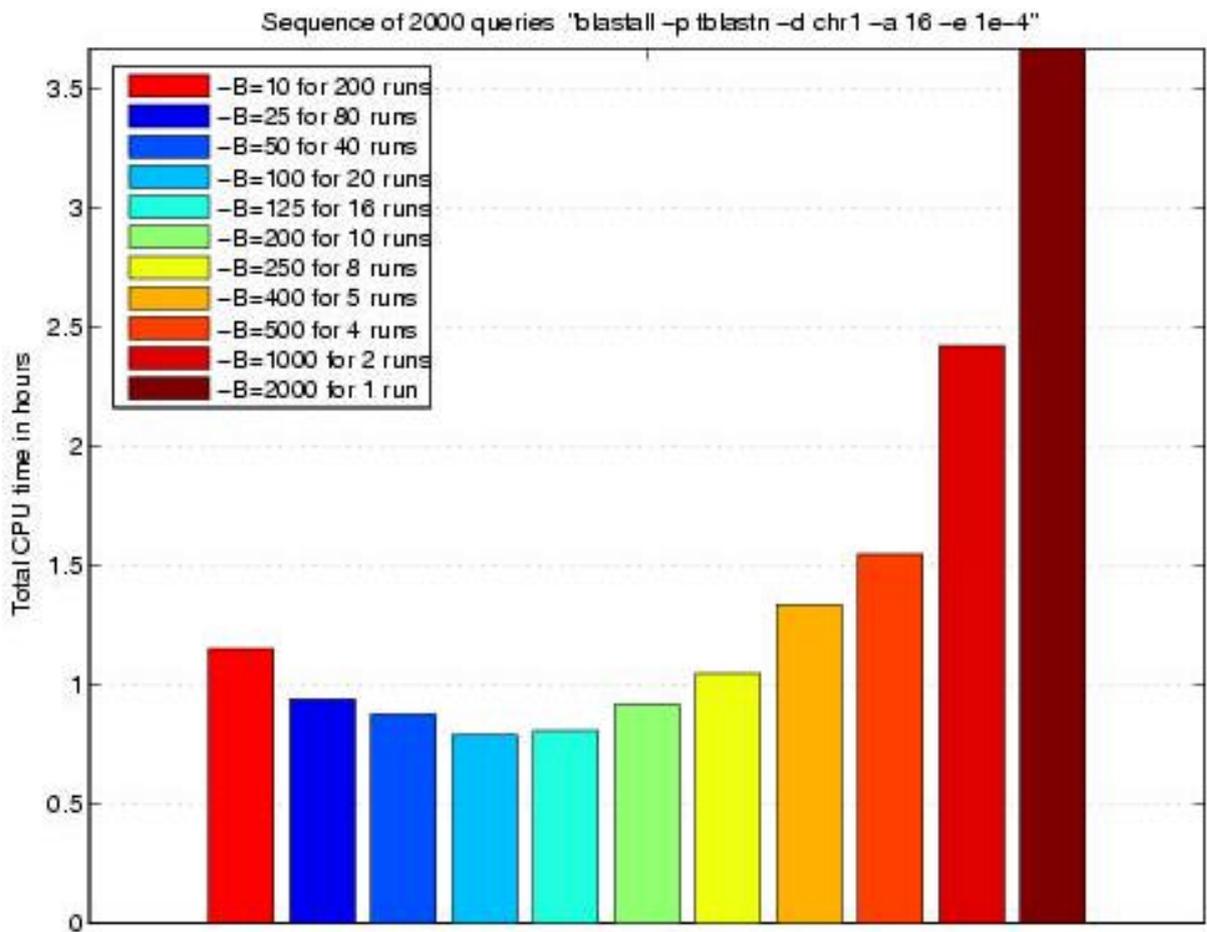
The **-B** option allows concatenating several queries in order to optimize the blast search. Indeed, the query length has a strong influence on the CPU time and it is important to find the optimal length for the query.

To test the influence of the **-B** option, we used as query a file of 2000 protein sequences, different values for the **-B** option and as database the first chromosome of the human genome. That blast was run using 16 threads:

```
blastall -p tblastn -d chr1 -a 16 -e 1e-4
```

- ? -B=10, we do 200 runs with blast
- ? -B=25, we do 80 runs with blast
- ? -B=50, we do 40 runs with blast
- ? -B=100, we do 20 runs with blast
- ? -B=125, we do 16 runs with blast
- ? -B=200, we do 10 runs with blast

- ? -B=250, we do 8 runs with blast
- ? -B=400, we do 5 runs with blast
- ? -B=500, we do 4 runs with blast
- ? -B=1000, we do 2 runs with blast
- ? -B=2000, we do 1 runs with blast



Looking at the results it is obvious that the optimal value for \verb+-B+ option seems to lie between 100 and 250. It is also necessary to find a compromise between the choice of the value for \verb+-B+ option and the number of output files created. This is why the value for \verb+-B+ option will be fixed at 256.

Annex A: Limitation on the number of concatenated queries**1. ncbi/include/blastconcatdef.h (line 84):**

```
typedef struct queries {
/* IDRIS modification 22/02/05
   NumQueries is now declared as Uint4 instead of Uint1 */
   Uint4 NumQueries;
}
```

2. ncbi/include/blastconcatdef.h (line 146):

```
/* IDRIS modification 22/02/05 for the two following prototype
   BlastMakeFakeBspConcat and BlastMakeMultQueries */

/* BlastMakeFakeBspConcat: Uint1 num_bsps
   is replaced by Uint4 num_bsps */
BioseqPtr LIBCALL BlastMakeFakeBspConcat
PROTO((BspArray bsp_arr, Uint4 num_bsps, Boolean is_na,
        Uint4 num_spacers));

/* Uint1 num_queries is replaced by Uint4 num_queries */
QueriesPtr LIBCALL BlastMakeMultQueries
PROTO((BspArray fbsp_arr, Uint4 num_queries, Boolean is_na,
        Uint1 num_spacers, SeqLocPtr PNTR lcase_mask_arr ));
/* END IDRIS modification */
```

3. C/demo/blastall.c+ (line 850):

```
/* IDRIS modification Uint1 is replaced by Uint4
   for the following variables 22/02/05 */
   Uint4 num_queries;
   Uint4 num_iters;
   Uint4 sap_iter;
/* END IDRIS modification */
```

4. C/demo/blastall.c (line 859):

```
/* IDRIS modification Uint1 is replaced by Uint4
   for the following variable 22/02/05 */
   Uint4 bsp_iter;
/* END IDRIS modification */
```

5. tools/blastconcat.c+ (line 622):

```
/* IDRIS modification 22/02/05
   Uint1 num_bsps is replaced by
   Uint4 num_bsps */
BioseqPtr LIBCALL
BlastMakeFakeBspConcat PROTO((BspArray bsp_arr, Uint4 num_bsps,
                               Boolean is_na, Uint4 num_spacers))
```

6. tools/blastconcat.c+ (line 833):

```
/* IDRIS modification 22/02/05 Uint1 num_queries
   is replaced by Uint4 num_queries */
QueriesPtr LIBCALL
BlastMakeMultQueries PROTO((BspArray bsp_arr, Uint4 num_queries,
                             Boolean is_na, Uint1 spacer_len,
                             SeqLocPtr PNTR lcase_mask_arr)) {
```

Annex B: BLAST report Output**1. ncbi/demo/blastall.c (line 1412):**

```

/* IDRIS 24/02/05 : modification of the blast report output with -m 0
                    queries are not all written before any computations
                    i.e. the following paragraph is commented */
/*if (align_view != 0)
    if (!options->is_megablast_search) {
        /* KM added loop here for concat case */
        num_iters = (num_queries>0) ? num_queries : 1;
        for (bsp_iter=0; bsp_iter<num_iters; bsp_iter++) {
            curr_bsp = (num_queries>0) ?
                *(fake_bsp_arr + bsp_iter) :
                query_bsp;
            AcknowledgeBlastQuery(curr_bsp, 70, outfp, believe_query, html);
        }
    }
*/

```

2. ncbi/demo/blastall.c+ (line 1580):

```

if(align_view < 7) {
    /* IDRIS modification22/02/05 : add \n */
    fprintf(global_fp, "%s\n", "done");
}

```

3. ncbi/demo/blastall.c+ (line 1711):

```

    } else if (align_view == 0) {
/* IDRIS modification: we add the following section for BLAST
   report output with -m option;
   there is now only one loop over all the queries
   so as aith the -B option we have:
   - output the corresponding query
   - loop over all the alignment sequences of this query
   to write the global alignment information
   - loop over all the alignment to write the details of
these
                    alignments
Note that if no alignments were found for a given query, nothing
is written (neither the query or alignments
*/
/* create the array of SeqAnnotPtrs, if necessary */

num_iters = (num_queries > 0) ? num_queries : 1;
for (sap_iter=0; sap_iter < num_iters; sap_iter++) {
    /* IDRIS modification : write the corresponding query */

```

```
curr_bsp = (num_queries>0) ? *(fake_bsp_arr + sap_iter) : query_bsp;
AcknowledgeBlastQuery(curr_bsp, 70, outfp, believe_query, html);

curr_seqalign = (num_queries > 0) ? *(sap_array + sap_iter) :
seqalign;
seqannot = SeqAnnotNew();
seqannot->type = 2;
AddAlignInfoToSeqAnnot(seqannot, align_type);
seqannot->data = curr_seqalign;
if (aip) {
    SeqAnnotAsnWrite((SeqAnnotPtr) seqannot, aip, NULL);
    AsnIoReset(aip);
}
curr_seqannot = seqannot;

if (outfp) { /* Uncacheing causes problems with ordinal nos. vs. gi's.
*/
/* IDRIS modification: if curr_seqalign is null then No hits were
found */
    if (curr_seqalign != NULL) {
        ObjMgrSetHold();
        /* print deflines */

        init_buff_ex(85);

        PrintDefLinesFromSeqAlignEx2(curr_seqalign, 80, outfp,
            print_options, FIRST_PASS, NULL,
            number_of_descriptions, NULL, NULL);

        free_buff();
        /* detail */

        /* AM: Query concatenation. */
        if( mult_queries && mask_loc )
        {
            orig_mask_loc = mask_loc;

            if( !mask_loc->data.ptrvalue ) mask_loc = NULL;
        }

        prune = BlastPruneHitsFromSeqAlign(curr_seqalign,
            number_of_alignments, NULL);
        curr_seqannot->data = prune->sap;

        if(options->is_ooframe) {
            OOFShowBlastAlignment(curr_seqalign, /*mask*/ NULL,
                outfp, align_options, txmatrix);
        } else {

            ShowTextAlignFromAnnot(curr_seqannot, 60, outfp, NULL, NULL,
                align_options, txmatrix, mask_loc,
                FormatScoreFunc);
        }

        curr_seqannot->data = curr_seqalign;
        prune = BlastPruneSapStructDestruct(prune);

        /* AM: Query concatenation. */
```

```
        if( mult_queries && orig_mask_loc )
        {
            mask_loc = orig_mask_loc;
            mask_loc = mask_loc->next;
        }
    /* END loop show text align, loop over seqalign/seqannots for
concat */
    ObjMgrClearHold();

    ObjMgrFreeCache(0);
}
} /* if outfp */
/* next alignment */

}
/*--KM free seqalign array and all seqaligns?? */
/* END IDRIS modification */
```

Annex C: Perl Script for FASTA Databases Segmentation

```
#!/usr/bin/perl -w
#
# FASTA Databases Segmentation
#
# Olivier Glorieux IDRIS 2004
#

use strict;
use Getopt::Long;

*TAILLE_LIGNE = \50 ;
my $ligneEntree ;
my $compteur ;
my $tailleMorceau = 0;
my $overlap = 0;
my @tableauOverlap ;

GetOptions(
    'tm|tailleDeMorceau:i' => \$tailleMorceau,
    'ov|overlap:i' => \$overlap
);

# checks if the chunk's size is positive
if ($tailleMorceau <= 0) {
    die("Usage splitSeqIdris.pl -tm int - ov int (the chunk's size
and the overlap \n the chunk's size cannot be zero or negative");
}

# checks if the value of the overlap is zero or positive
if ($overlap < 0) {
    die("the value of the overlap must be zero or positive");
}

# checks if the value of the overlap is smaller than the chunk's size
if ($overlap > $tailleMorceau) {
    die("the value of the overlap must be smaller than the chunk's
size");
}

# checks if the value of the overlap and the chunk's size are divisible
by the constant size of a fasta line (50)
if (($tailleMorceau % $main::TAILLE_LIGNE) != 0 || ($overlap %
$main::TAILLE_LIGNE) != 0) {
    die("the value of the overlap and the chunk's size must be
divisible by the constant size of a fasta line (50)");
}

while ($ligneEntree = <STDIN>) {

    # print("ligne stdin : ".$ligneEntree);

    # If the current line is the title of a sequence
```

```

        if (substr($ligneEntree,0,1)eq">") {
            $compteur = gerer($ligneEntree, $tailleMorceau,
$overlap) ;
            # else if it is part of the body of the sequence
        } else {
            # print("ligne : ".$ligneEntree);
            &$compteur($ligneEntree);
        }
    }
}

# that function manages the inputstream, intializes variables specific
to the
# sequence we are splitting. Then it returns a subfunction (closure)
that will
# actually split the sequence
sub gerer {
    my $titre = shift ;
    my $tailleMorceau = shift ;
    my $overlap = shift ;
    # stores the chunk's number of lines
    my $indexLigne = 0;
    # stores the chunk's maximum number of lines
    my $indexLigneMax = $tailleMorceau / $main::TAILLE_LIGNE ;
    # stores the number of characters of the sequence before
splitting
    my $indexSequence = 0 ;
    # stores the first line's index of what will be the overlap
    my $indexLigneOverlap = $indexLigneMax - $overlap /
$main::TAILLE_LIGNE ;
    # index to fill the array overlap
    my $indexOverlap = 0 ;
    # prints the title the first time
    printf(">0-%s ".substr ($titre,1),$tailleMorceau);
    $indexSequence += $tailleMorceau ;
    return sub {
        my $ligne = shift ;
        # prints the current line (in every case)
        print($ligne);
        # if we are in the overlap's zone
        #print("indexLigne : ".$indexLigne);
        if($indexLigne>=$indexLigneOverlap) {
            #print("bing\n");
            $tableauOverlap[$indexOverlap] = $ligne ;
            $indexOverlap++
        }
        $indexLigne++ ;
        #print("indexLigne : ".$indexLigne);
        # if we reached the chunk's size
        if ($indexLigne == $indexLigneMax) {
            # prints the new title
            printf(">%d-%d ".substr ($titre,1),
$indexSequence,$indexSequence+$tailleMorceau);
            #prints the array overlap
            afficherOverlap(*tableauOverlap) ;
            $indexSequence += $tailleMorceau ;
            $indexLigne = 0 ;
            $indexOverlap = 0 ;
        }
    }
}

```

```
}  
  
# that method allows us to print the data of the array overlap  
sub afficherOverlap {  
    (*tableauOverlap)=@_  
    my $i;  
    foreach $i(@tableauOverlap) {  
        print ($i);  
    }  
}
```

Annex D: Perl Script for BLAST Databases Update

```
#!/usr/bin/perl
#
# BLAST Databases Update
#
# Denis Raux IDRIS 2005
#

use strict;
use Net::FTP;
use Time::Local;
use FileHandle;

#local path
my $BIOPATH="/workdir/biogen";
my $DBPATH="$BIOPATH/db";
my $INFOBIOGENPATH="$DBPATH/infobiogen";
my $IDXPATH="$INFOBIOGENPATH/index-blast";
my $LOGPATH="$BIOPATH/log";
my $GOLDENPATH="$INFOBIOGENPATH/GoldenPath";
my $NCBIPATH="$DBPATH/ncbi/genomes/bacteria";
my $NCBIBASEFILE="$NCBIPATH/all.faa";
my $TMPDIR="$DBPATH/tmp";

#infobiogen path
my $BIOSERVER="ftp://babbage.infobiogen.fr/db";
my $BIOGOLDEN="$BIOSERVER/GoldenPath/GENOME_MIRROR/goldenPath";

#ncbi path
my $NCBIFAA="ftp://ftp.ncbi.nlm.nih.gov/genomes/Bacteria/*/*faa";

my $LISTNAME="$IDXPATH/.listing";
my $AGAINDELAY=600;
my $CMDBASE=~"/tools/wget -nv";
#my $CMDBASE=~"/tools/wget";
my $CMDOPT=$CMDBASE." -t 20 --waitretry=120 --connect-timeout=120 --
read-timeout=120";

#definitions of files to be loaded
my @GOLDENFILE=qw(panTrol canFam1 mm5 rn3 galGal2 danRer2 tetNig1
droYak1 anoGam1 ce2 hg17);

my ($progname)=$0=~/(^[^/]+)$/;

my @candidates=("zahir001","zahir009"); #Hosts capable to run

$SIG{KILL} = sub {exit;};
$SIG{TERM} = $SIG{KILL};
$SIG{QUIT} = $SIG{KILL};
$SIG{ABRT} = $SIG{KILL};
$SIG{INT} = $SIG{KILL};
```

```

sub max($$) {$_[0] > $_[1]?$_[0]:$_[1]}
sub min($$) {$_[0] < $_[1]?$_[0]:$_[1]}

sub Gm($) {
  my $fct=shift;
  my $f=sub($$){
    my $n1=shift;
    my $n2=shift;

    if(defined $n1 && defined $n2) {
      $n1=&$fct($n1,$n2);
      foreach(@_) {$n1=&$fct($n1,$_);}
    }
    return $n1;
  };
  return $f;
}

*Min=Gm(\&min);
*Max=Gm(\&max);

{
  package d_Lock;
  use FileHandle;

  sub new($) {
    my $this={'name'=>shift(@_).".lock"};
    my $fh = new FileHandle($this->{'name'}, "r");
    if(defined $fh) {
      my $pid=$fh->getline;
      return undef if defined $pid && !system("/usr/bin/ps -p
$pid>/dev/null") && !$?;
      $fh->close;
    }
    $fh = new FileHandle($this->{'name'}, "w");
    if(defined $fh) {
      $fh->print("$$$$");
      $fh->close;
    } else {undef $this->{name};}
    bless $this;
  }
  sub DESTROY {
    my $this=shift;
    unlink($this->{'name'}) if defined $this->{'name'};
  }
}

{ package FaaFile;
  @FaaFile::ISA=("FileHandle");

  my $BaseName="$NCBIPATH/All.faa";
  my $count=0;

  sub new() {
    $count++;
    my $fh=FileHandle::new("FaaFile");
    my $file=sprintf("%s%.4d",$BaseName,$count);
    $fh->FileHandle::open($file,">");

```

```

        return $fh;
    }
    sub DESTROY {
        my $this=shift;
        $this->FileHandle::close();
    }
}

sub tabcomp(\@@) {
    my $t1=shift;
    my @s=@_;
    foreach (@$t1) {
        my $i=shift @_;
        next if (/^\d+$/ && $i=~/\d+$/ && $_ == $i) || $_ eq $i;
        return 1;
    }
    return(0);
}

sub logcount($) {
    my $log=shift;my $KBcount=0;my $filecount=0;my $maxretry=0;
    foreach (<$log>) {
        my ($c,$i)=/[(\d+)\].*?[(\d+)\]/;
        next if !defined $c || !defined $i;
        $KBcount+=$c;
        $filecount++;
        $maxretry=max($i,$maxretry);
    }
    return($filecount,$KBcount/1024,$maxretry);
}

sub GetGrId($) {
    my $gr=shift;
    my $id;
    open(GRP,'<','/etc/group');
    while(<GRP>) {
        ($id)=/$gr:!(\d+)/;
        last if defined $id;
    }
    close GRP;
    return $id;
}

sub FileTime($) {
    my $file=shift;
    my
    (undef,undef,undef,undef,undef,undef,undef,undef,undef,$mtime)=stat($file);
    return $mtime;
}

sub GetDirTime($) {
    my $dir=shift;
    my @ret;
    return undef if !opendir DIR,$dir;
    foreach(readdir DIR) {next if
/^\.\.?$/;push(@ret,FileTime("$dir/$_"));}
    closedir DIR;
    return @ret;
}

```

```

}

sub ParseAdd($) {
    my $path = shift;
    return $path=~ftp:\/\\[^\[/]+\)\/(.+)$/;
}

sub GetFtpTime($) {
    my $path=shift;

    my (undef,undef,undef,undef,undef,$CurYear)=localtime;
    my
    %mon=("Jan",0,"Feb",1,"Mar",2,"Apr",3,"May",4,"Jun",5,"Jul",6,"Aug",7,"S
ep",8,"Oct",9,"Nov",10,"Dec",11);
    my @Times;

    my ($host,$dir)=ParseAdd($path);

    if(defined $host && defined $dir) {
        my $ftp = Net::FTP->new($host, Debug => 0) or return undef;
        $ftp->login("anonymous","anonymous") or return undef;
        foreach ($ftp->dir($dir)) {
            my
            (undef,undef,undef,undef,undef,$month,$mday,$year)=split;
            next if !defined $year;
            my ($hour,$min)=$year~/(\d+):(\d+)/;
            $year = defined $hour?$CurYear:$year-1900;
            $hour=0 if !defined $hour;
            $min=0 if !defined $min;
            push @Times,
            timelocal(0,$min,$hour,$mday,$mon{$month},$year);
        }
        $ftp->quit;
    }
    return(@Times);
}

sub MkTree($$) {
    my $path=shift; #start path
    my $fullpath=shift; #full path
    while(defined $path && $fullpath ne $path) {
        ($path)=$fullpath=~/^($path\[^\[/]+)/;
        last if defined $path && !-e $path && !mkdir $path;
    }
    return -e $fullpath?1:0;
}

{ package Myftp;

    @Myftp::ISA=("Net::FTP");
    my $user;

    sub new($) {
        my $path=shift;
        my ($host,$dir)=main::ParseAdd($NCBIFAA);
        my $ftp= Net::FTP::new("Myftp",$host, Timeout=>20);

```

```

        return undef if !defined $ftp || !$ftp-
>login("anonymous","anonymous");
    $ftp->binary();
    ${*$ftp}->{__PACKAGE__ ."host"}=$host;
    ${*$ftp}->{__PACKAGE__ ."dir"}=$dir;
    return $ftp;
}
sub ls {my $this=shift;return $this->Net::FTP::ls(${*$this}-
>{__PACKAGE__ ."dir"});}
sub open($) {
    my $ftp=shift;
    my $file=shift;
    delete ${*$ftp}{'net_ftp_port'};
    delete ${*$ftp}{'net_ftp_pasv'};
    my $data = $ftp->retr($file) or return undef;
    return($data,${*$ftp}{'net_ftp_blksize'});
}
}

sub FaaWrite() {
    my $fh=FaaFile::new();
    my $count=0;
    my $nl=1; #new line just before
    return sub(\$) {
        my $str=shift;
        my $offset=-1;
        while(($offset=index($str,">",$offset+1)) >=0) {
            if($offset) {next if substr($str,$offset-1,1) ne "\n"}
else {next if !$nl};
            if(++$count>256) {
                return 0 if !$fh->print(substr($str,0,$offset));
                $fh=FaaFile::new();$str =\substr($str,$offset);
                $offset=0;$count=1;
            }
        }
        $nl=substr($str,length($str)-1,1) eq "\n"?1:0;
        return($fh->print($str));
    }
}

sub DelDir {
    my $dir=shift;
    unlink <$dir/*>;
    foreach (<$dir/*>) {
        my $ret=DelDir($_);
        return $ret if $ret;
    }
    return rmdir($dir);
}

sub goldenpath {
    # The only way to reconstruct is to remove the $GOLDENPATH file.
    if(!MkTree($DBPATH,$GOLDENPATH)) {
        warn("can't create $GOLDENPATH\n");
        return 1;
    }
    DelDir($TMPDIR);
    if(!mkdir($TMPDIR)) {
        warn("can't create $TMPDIR\n");

```

```

        return 1;
    }
    foreach(@GOLDENFILE) {
        my $lfile="$GOLDENPATH/$_";
        my $rfile="$BIOGOLDEN/$_/chromosomes/*fa.*";

        my @rtime=GetFtpTime($rfile);
        if(!@rtime) {warn("can't access file : $rfile\n");next;}
        my $Rtime=Max(@rtime);
        next if -e $lfile && FileTime($lfile)>=$Rtime;

        if(system "$CMDOPT -q -r -nd -A fa.zip,fa.gz -P $TMPDIR
$BIOGOLDEN/$_/chromosomes" || $? ) {
            warn("wget error on $BIOGOLDEN/$_/chromosomes fa.gz\n");
        } else {
            system('/usr/bin/gunzip -c -S "" '."$TMPDIR/* |
~/bioinfo/splitSeqIdris.pl -tm 500000 -ov 0 | ~/bioinfo/formatdb -p F -i
stdin -o T -n $GOLDENPATH/$_");
            utime $Rtime,$Rtime,<$GOLDENPATH/$_*>;
            unlink(<$TMPDIR/*>);
        }
    }
    DelDir($TMPDIR);
    return 0;
}

sub ncbi {
    if(!MkTree($DBPATH,$NCBIPATH)) {
        warn("can't create $NCBIPATH\n");
        return 1;
    }
    my @rtime=GetFtpTime($NCBIFAA);
    if(!@rtime) {warn("can't access file :$NCBIFAA \n");return 1;}
    my @ltime=GetDirTime($NCBIPATH);
    my $Rtime=Max(@rtime);
    return 0 if @ltime && Min(@ltime) >= $Rtime;

    my $ftp=Myftp->new($NCBIFAA);
    if(!$ftp) {warn("can't connect $NCBIFAA\n");return 1;}
    my $faa=FaaWrite();

    foreach ($ftp->ls) {
        my ($file,$size)=$ftp->open($_);
        next if !defined $file;
        my $buf;
        while($file->read($buf,$size) {
            if(!&$faa(\$buf)) {
                warn("Error writing in Faa files\n");
                return(1);
            }
        }
        $file->close();
    }
    utime $Rtime,$Rtime,<$NCBIPATH/*>;
    return 0;
}

my $hostname=`hostname`;

```

```

die "/etc/nodeinter not open on host $hostname" if !defined
open(INTER,'<','/etc/nodeinter');
foreach (@candidates) {
    exit if `hostname` eq readline(*INTER); # mustn't run on interactiv
}
close INTER;

my $lock=d_Lock::new("$LOGPATH/$progname");
die "Already running" if !defined $lock;

umask(027);
goldenpath;
ncbi;

if(!MkTree($DBPATH,$IDXPATH)) {
    warn("can't create, $IDXPATH\n");
    return 1;
}
my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) =
localtime(time);
my $logname="$LOGPATH/wget_`$wday`.log";

my $mode="o";my $downloadok=0;

#open(LOG,'+<',$logname) ||
open(LOG,'+>',$logname);
while(!$downloadok) {
    my $ret=0;$downloadok=1;
    $ret=system("$CMDOPT -N -P $IDXPATH -`$mode` $logname $BIOSEVER/index-
blast/*");
    $mode="a";

    ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) =
localtime(time);
    printf(LOG "%2.2d:%2.2d transfert %s stop with code/signal %d/%d.
files:%d KBytes:%u
MaxRetry:%d\n",$hour,$min,$IDXPATH,$ret>>8,$ret&l27,logcount(*LOG));
    last if $ret;

    $ret=system("$CMDBASE." -nr -P $IDXPATH $BIOSEVER/index-
blast/*getlisting* 2>/dev/null");
    if($ret || !defined open(LST,'<',$LISTNAME)) {
        print(LOG "listing not found, processing aborted\n");
        $downloadok=0;
    } else {
        my %remote;undef %remote;keys(%remote)=4096;
        foreach(<LST>) {

my($size,$month,$day,$time,$name)=/(\d+)\s+(\w+)\s+(\d\d?)\s+(\d\d?:?\d\d
)\s+(.*)$/;
        @{$remote{$name}}=($size,$month,$day,$time) if defined
$name;
        }
        delete $remote{'.listing'}; #I've already seen it on babbage
        #print scalar(%remote)," ", scalar keys %remote ;exit;
        close LST;

```

```

        unlink($LISTNAME);
        my @local=`ls -l $IDXPATH`;
        if((keys %remote) < (@local - int(0.2 * @local))) {
            printf(LOG "Local:%d Remote:%d. Remote list probably
corupt...try again\n",scalar(keys %remote), scalar(@local));
            $downloadok=0;
        } else {
            foreach(@local) {
                # pattern $size,$month,$day,$time,$name

my(@item)=/(\d+)\s+(\w+)\s+(\d\d?)\s+(\d\d?:?\d\d)\s+(.*)$/;
                next if @item != 5;
                my $name=$item[4];
                if (!defined $remote{$name}) {
                    print(LOG "Remote not found =>delete @item\n");
                    unlink("$IDXPATH/$name");
                } else {
                    if (tabcomp(@{$remote{$name}},@item)) {
                        print(LOG "mismatch remote
@{$remote{$name}}=>delete @item\n");
                        $downloadok=0;
                        unlink("$IDXPATH/$name");
                    }
                    delete $remote{$name};
                }
            }
        }
        if(%remote) {
            foreach (keys %remote) { print(LOG "Remote file only
$_ @{$remote{$_}}\n");
                $downloadok=0;
                print(LOG "Download again\n");
            }
        }
    }
    my $time0=time;
    system("$CMDOPT -N -xP$INFOBIOGENPATH -nH --cut-dirs=1 -a$logname
$BIOSERVER/NR/NRnuc/NucAll $BIOSERVER/NR/NRprot/ProtAll");
    seek LOG,0,2;
    if(!$downloadok) {
        my $nextdl= time>$time0+$AGAINDELAY?0:$AGAINDELAY;
        printf(LOG " ***** Next download in %d seconds
*****\n", $nextdl);
        sleep($nextdl) if $nextdl ;
    }
}
seek LOG,0,0;
printf(LOG "%2.2d:%2.2d End transfert. files:%d KBytes:%u
MaxRetry:%d\n", $hour, $min, logcount(*LOG));

```

