Journal of Bioinformatics and Computational Biology
Vol. 12, No. 3 (2014) 1450013 (17 pages)
© Imperial College Press
DOI: 10.1142/S0219720014500139

Imperial College Press
www.icpress.co.uk

MPI-blastn and NCBI-TaxCollector: Improving metagenomic analysis with high performance classification and wide taxonomic attachment

R. Dias[*,§], M. G. Xavier[†], F. D. Rossi[†], M. V. Neves[†], T. A. P. Lange[†],
A. Giongo[‡], C. A. F. De Rose[†] and E. W. Triplett[*]

[*]*Department of Microbiology and Cell Science*
University of Florida, Florida, United States

[†]*Faculty of Informatics*
Pontiphical Catholic University of Rio Grande do Sul
Porto Alegre/RS – Brazil

[‡]*Faculty of Biosciences*
Pontiphical Catholic University of Rio Grande do Sul
Porto Alegre/RS – Brazil
[§]*raquel.dias@ufl.edu*

Received 15 October 2013
Revised 11 March 2014
Accepted 12 March 2014
Published 24 June 2014

Metagenomic sequencing technologies are advancing rapidly and the size of output data from high-throughput genetic sequencing has increased substantially over the years. This brings us to a scenario where advanced computational optimizations are requested to perform a metagenomic analysis. In this paper, we describe a new parallel implementation of nucleotide BLAST (MPI-blastn) and a new tool for taxonomic attachment of Basic Local Alignment Search Tool (BLAST) results that supports the NCBI taxonomy (NCBI-TaxCollector). MPI-blastn obtained a high performance when compared to the mpiBLAST and ScalaBLAST. In our best case, MPI-blastn was able to run 408 times faster in 384 cores. Our evaluations demonstrated that NCBI-TaxCollector is able to perform taxonomic attachments 125 times faster and needs 120 times less RAM than the previous TaxCollector. Through our optimizations, a multiple sequence search that currently takes 37 hours can be performed in less than 6 min and a post processing with NCBI taxonomic data attachment, which takes 48 hours, now is able to run in 23 min.

Keywords: BLAST; TaxCollector; NCBI; sequence alignment; metagenomics; taxonomy assignment; taxonomic attachment.

1. Background

Our planet is populated by a large number of microbial cells (about 10^{30}),[1] most of them still remain uncultivable.[2] New sequencing technologies have allowed the studies on diversity and abundance of environmental microbiota. The data obtained from metagenomic studies provides information about the structure, organization, evolution, and origin of the organisms.[3]

In the last few years, sequencing technologies have rapidly advanced and the size of output data from high-throughput genetic sequencing has increased substantially. Next-generation sequencing has improved sequencing speed from 10 Mb per day to near 120 Gb per day.[4,5] A metagenomic analysis consists of a multi-step procedure that filters the reads by quality, searches the filtered sequences within a genetic database with the goal of finding sequence matches, assigns a taxonomic classification on the sequence matches, performs statistical analysis on classified and unclassified sequences, summarizing and visualizing the results. Some of these steps must be performed by using large database resources, such as National Center for Biotechnology Information advances (NCBI),[6] Greengenes,[7] and RDP[8] databases, as well as a large number of input sequences. Therefore, stand-alone and online tools were developed and used to perform faster and more automated analysis. Some examples of these tools are pipelines for analysis of next generation amplicons (PANGEA),[9] Mothur,[10] and Ribosomal Database Project (RDP) classifier.[11] These tools, known as metagenomic pipelines, optimize the runtime of high-throughput sequencing analysis by trying to minimize the human intervention on this process.

Despite computational tools having improved the metagenomic studies, the amount of sequences generated by next generation technologies are increasing at an even higher speed. Studies by Kahn *et al.* (2010) show that a doubling of sequencing output every 9 months has surpassed the performance improvements of disk storage and high-performance computation fields.[4] Therefore, we are closer to a scenario where advanced computational optimizations will be strictly necessary to conclude an ordinary metagenomic analysis. Considering this, our optimizations and performance evaluations are focused in some of the most complex and time-consuming steps of a metagenomic analysis: Taxonomic classification assignment and post processing of classification results. In the next section, we review the current methods used in taxonomic classification and the main strategies applied for performance improvement.

1.1. *Taxonomic classification assignment with BLAST*

The first step in a metagenomic analysis involves the comparison of several sequences to known sequence databases, using tools such as Basic Local Alignment Search Tool (BLAST).[12] This step, known as taxonomic classification or annotation, is a computationally heavy task and has been the target of several computational optimization studies.[13–15]

BLAST is still one of the most used tools for taxonomic classification in the first (and critical) classification steps of metagenomic analyses, proving to be successful in many studies.[16,17] It is possible to run an online BLAST sequence search from the NCBI Web-server.[6] However, the NCBI BLAST Web-server does not support the submission of multiple input sequences simultaneously, neither the use of external databases such as Greengenes,[7] RDP[18] or local resources. To overcome the online limitations of BLAST its stand-alone version is used, enabling local analyses of new metagenomic datasets and comparisons of datasets from different environments. In addition, many optimizations of high performance computing were implemented for improving BLAST algorithm. The BLAST+ version of BLAST implements lower complexity optimizations and multi-threading support.[19] However, BLAST+ is still unable to run in more than one node of a computer cluster at the same time, limiting scalability of this tool in such environments.

To handle BLAST limitations, distributed versions of BLAST were developed, such as mpiBLAST[20] and ScalaBLAST.[21] mpiBLAST improved the performance of the computationally intensive sequence alignment process as more nodes can be used. The parallelization strategy of mpiBLAST is based on the partitioning of the input database into many fragments, as many as the number of nodes to be executed.[20] In the same way, ScalaBLAST partitions the input queries into fragments. However, ScalaBLAST keeps separate copies of the database and query list in RAM on each core and the rest of the RAM is reserved for BLASTs compute kernel, which can be memory consuming. In both cases, these input query fragments are copied to each node and a local search is performed.[21] However, ScalaBLAST keeps separate copies of the database and query list in RAM on each core and the rest of the RAM is used for BLASTs compute kernel. After the parallel BLAST search steps, the results are merged among processes. The input partitioning and the following steps may generate working overhead, when dealing with large databases, such as the NCBIs nt/nr, or large queries such as the next-generation sequencing outputs. Furthermore, a parallel search that partitions the database needs to combine sequence segments found from individual database partitions for each query sequence, requiring extra post processing if compared to sequential BLAST. Recently, GNU parallel has been used as a simple solution for running multiple BLAST searches in one or more nodes.[22] GNU parallel is a shell tool that can split a given input and divide it into multiple commands. For running BLAST among several nodes, GNU parallel uses multiple Secure Shell (SSH) connections. However, cryptographic methods are central to SSH and these methods may affect performance due to encryption overhead.[23] In the present work, we implement a novel strategy of parallelization for BLAST, focusing on nucleotide alignment function (MPI-blastn). The present tool is available at https://github.com/Bioinfo-Tools/MPI-blastn.

1.2. *Post processing of BLAST results with TaxCollector*

The typical BLAST result in a metagenomic analysis consists of a text file in a tabular format with information on subject and query alignment, such as name of the subject and query sequence, similarity score, e-value, bitscore, and sequence coverage. Usually, the query name provides no information on its taxonomic classification levels. For obtaining this information, the user must search the query identification number (TAXID or GI number) in taxonomy databases, such as NCBI,[6] Greengenes[7] or RDP taxonomy.[18]

Considering that a metagenomic analysis may have hundreds of thousands of input sequences, the manual taxonomic ranks assignment becomes impracticable. TaxCollector tool was developed by Giongo *et al.*[24] in order to handle this problem, attaching taxonomic information on 16S rRNA sequences from RDP and Greengenes databases. TaxCollector approach involves the loading of taxonomy databases from NCBI (names.dmp and nodes.dmp files from taxdump.tar.gz databases[25]) and their conversion to dictionary structures in Python language. All the taxonomic databases are loaded into computer RAM (about 4 GB). However, loading large databases into RAM does not seem to be the most promising approach. Another limitation is that TaxCollector scripts support only RDP and Greengenes databases, not handling with the largest genetic database available, NCBI nt/nr databases.[6] Including the NCBI database implies the attachment of a new database file in the algorithm. This new database, known as GenInfo Identifier (GI) versus taxon ID (gi_taxid_nucl.dmp) is available at the NCBI Taxonomy database and it has GI identification numbers, from GenBank,[26] converted to their corresponding TAXID numbers, supported by NCBIs taxonomic trees (nodes.dmp) and names (names.dmp) database files.[25] Considering TaxCollector limitations, we propose a new algorithm of lower complexity and higher performance for taxonomic levels attachment of the NCBI database. The new algorithm, NCBI-TaxCollector, is described and compared to TaxCollector. The present tool is available at https://github.com/Bioinfo-Tools/NCBI-taxcollector.

2. Implementation

In this section, we describe the implementation of the tools proposed in the present work. Figure 1 demonstrates a summary of the main steps of our algorithms (dashed rectangles), as well as their placements in a metagenomic analysis context. As mentioned in a previous section, our optimizations are applied in some of the most complex and time consuming steps in the post processing of metagenomic data: Species classification and post-processing with taxonomic attachment (relative time represented in the runtime axis in the bottom of Fig. 1). More details on the algorithms implemented for such tasks are discussed in the next sections.

2.1. *MPI-blastn: Parallel nucleotide search for computing clusters*

MPI-blastn is a new parallel implementation based on a recent version of BLAST+ (2.2.25). Unlike BLAST+, which can only take advantage of individual shared-memory multicore machines, our implementation exploits the computing power of clusters of multicore machines, to allow increased performance and scalability. Our approach consists of two steps. In the first step, the algorithm splits the input queries evenly among the number of available computer nodes and places these sub-queries on shared storage together with the NCBI database. In the second step, each node loads its sub-query and a copy of the full database to local memory, and the parallel execution is started. If multiple cores are available in a node, all take part in the calculations. Message Passing Interface (MPI) is used to exchange messages among

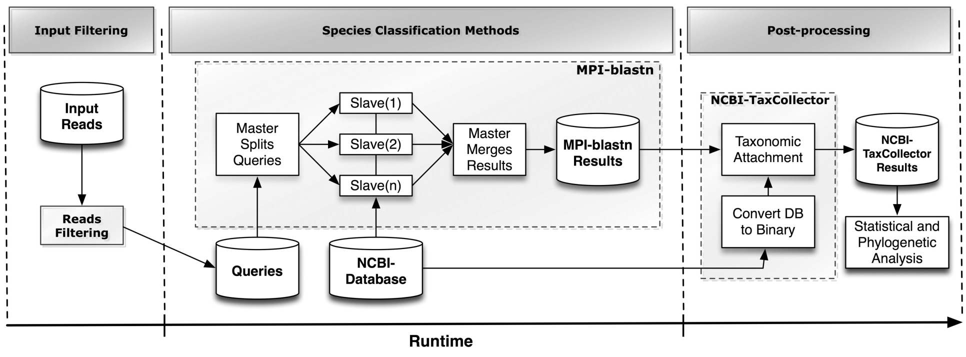
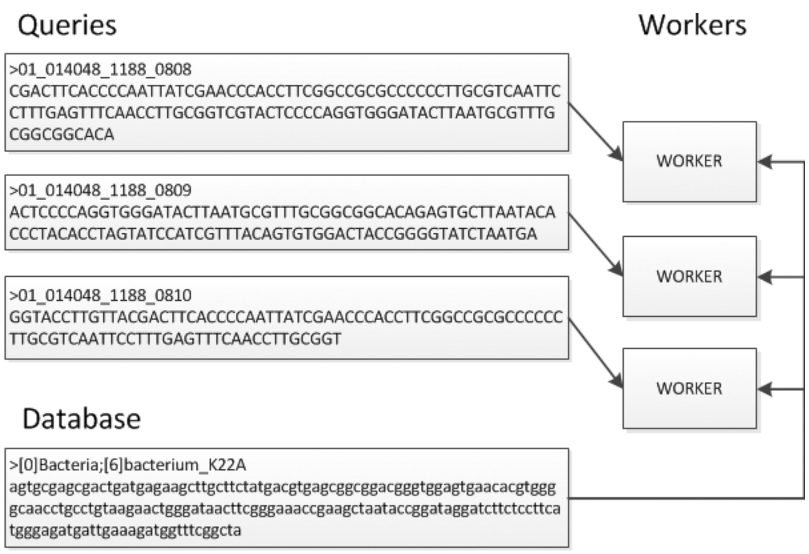


Fig. 1. Overall workflow of proposed optimizations for metagenomic analysis.

the computing nodes of the cluster through a master-slave programing strategy, and threads are used inside the slaves to explore the multiple cores of a node. The master process is responsible for step 1 and all other processes are slaves, executing step 2.

When the master process starts, each worker waits until the master splits all the input queries. After splitting the queries, the master process reports to all workers a start signal, indicating that the split stage was finished. After that, workers load the queries and the database from shared storage and run the computation of all their respective queries (Fig. 2). When the computation is done, all workers report to the master process that they are done. After performing the sequence search, the output of MPI-blastn is a set of result files from every worker. Lastly, the master process merges all search results obtained from workers, generating a single output file.



```
>01_014048_1188_0808
CGACTTCACCCCAATTATCGAACCCACCTTCGGCCGCGCCCCCCTTGCGTCAATTC
CTTTGAGTTTCAACCTTGCGGTCGTACTCCCCAGGTGGGATACTTAATGCGTTTG
CGGCGGCACA
```

```
>01_014048_1188_0809
ACTCCCCAGGTGGGATACTTAATGCGTTTGCGGCGGCACAGAGTGCTTAATACA
CCCTACACCTAGTATCCATCGTTTACAGTGTGGACTACCGGGGTATCTAATGA
```

```
>01_014048_1188_0810
GGTACCTTGTTACGACTTCACCCCAATTATCGAACCCACCTTCGGCCGCGCCCCCC
TTGCGTCAATTCCTTTGAGTTTCAACCTTGCGGT
```

```
>[0]Bacteria;[6]bacterium_K22A
agtgcgagcgactgatgagaagcttgcttctatgacgtgagcggcggacgggtggagtgaacacgtggg
gcaacctgcctgtaagaactgggataacttcgggaaaccgaagctaataccggataggatcttctccttca
tgggagatgattgaaagatggtttcggcta
```

Fig. 2. Input data of MPI-blastn algorithm in detail.

Our parallel implementation shows three main advantages if compared to a common parallel script: In our master-slave programing strategy (1) the workload is split and distributed evenly among all processors; (2) the master processors monitors all slave processors, certifying that all slaves accomplish their tasks correctly; and (3) once all tasks are performed, the output data is merged automatically. In our implementation, MPI messages do not carry the actual input data and are used just to make sure the synchronism between workers and the master node. This decision was made to simplify the implementation reducing the message size (depending on the problem size, queries may have up to 8 MB and the database used is about 4 GB big). This has a minimal impact in the performance, because the time needed to split and merge the queries is very small compared with the total time of the computations. In the worst case (small workloads), this I/O represents only 3% of the total execution time, as demonstrated in Refs. 21 and 27.

2.2. *NCBI-TaxCollector*

As mentioned above, the programming approach of TaxCollector developed by Giongo *et al.*[24] involves the loading of NCBI Taxonomy databases and their conversion to dictionary structures in Python programming language. Whole taxonomic databases are loaded into RAM (about 4 GB of size), GI identifications from BLAST results are translated to NCBI taxon IDs, and taxonomic information is attached to them. However, considering that these databases are updated often, increasing their sizes, loading databases into RAM does not seem to be the most suitable approach in this case.

Our NCBI-TaxCollector tool has a new approach to load the GI number from BLAST results and to search their information on taxonomic rank levels from NCBI databases. This tool is designed to maximize performance and allows many searches per second. NCBI-TaxCollector runs as a command line tool and, for this reason, it is especially suitable for use in scripts.

This tool takes as input a GI value from BLAST search results and the NCBI database files (namely gi_taxid_nucl.dmp, nodes.dmp and names.dmp). The file gi_taxid_nucl.dmp maps the GI numbers to their corresponding NCBI taxon IDs. The file nodes.dmp has a child–parent structure, which is responsible for modeling the different taxonomic levels. The file names.dmp contains a list with one or more taxonomic names for each NCBI taxon ID. Thus, the algorithm consists of four steps: (1) translating the GI value to its corresponding NCBI taxon ID; (2) traversing the nodes list looking for the highest parent node; (3) finding all available taxonomic names for each taxonomic level; and then, (4) returning this information in a human-readable format.

First, the tool converts the NCBI databases, originally in ASCII format, to a binary format optimized for fast searches. This optimization consists of generating a sorted list of NCBI taxon ID values, whereas each NCBI taxon ID is stored in one line index that is equal to the number of its corresponding GI value. For example, if we want to know what is the NCBI taxon ID number n that refers to GI i, the program goes to the line *i*. This file structure allows the NCBI-TaxCollector to find an NCBI taxon ID through one unique search step, seeking directly the line index that stores the taxonomic information. Combined to this optimized structure, the convertion to binary format not only speeds up the search, but also reduces the NCBI file size. For example, we observed up to 54% reduction on the total size of the bases in our experiments.

Once the optimized database is generated, the translation of a GI value to a NCBI ID is performed with a very small computational complexity, known as O(1) in algorithmic notation,[28] i.e. it is independent of the number of NCBI entries. Since the size of these databases is constantly increasing, this is a key feature for taxonomic tools. On the other hand, the databases for mapping taxonomic names (nodes.dmp and names.dmp) have a hierarchical structure, which cannot be optimized using the same approach as the gi_taxid_nucl.dmp database. Since each node in this hierarchical structure may refer to other nodes and different taxonomic names, we cannot optimize the binary file structure so each taxonomical node has an unique reference value. In order to minimize this problem, the search for taxonomic names is performed using a binary search approach that has a higher computational complexity of O(log n), where n is the number of entries in the database. The final output of NCBI-TaxCollector is the BLAST result initially provided by the user with all GI values replaced by their corresponding taxonomical classification in six levels, from domain to species, in a human-readable format. Table 1 describes an example of input and output of NCBI-TaxCollector.

Table 1. Examples of input and output for NCBI-TaxCollector.

Input (Blast/MPI-Blastn results)
Query1 gi||309261160|*gb*|*HQ*246245.1|
81.87 1186 148 64 226 1375 128 1282 0.0 937
Query2 gi||85001879|*gb*|*DQ*337061.1|
79.47 1554 166 122 9 1464 1 1499 0.0 961
Query3 gi||319992851|*emb*|*FR*729081.1|
82.33 928 124 40 446 1352 115 1023 0.0 769
Query4 gi||309261157|*gb*|*HQ*246242.1|
81.97 976 112 61 428 1371 401 1344 0.0 769
Output (Blast results modified by NCBI-TaxCollector)
Query1 [0]Bacteria; [1]Proteobacteria; [2]Alphaproteobacteria; [3]Rhodospirillales; [4]Acetobacteraceae; [5]Roseomonas; [6]Roseomonas_sp.;
81.87 1186 148 64 226 1375 128 1282 0.0 937
Query2 [0]Bacteria; [5]uncultured_bacterium; [6]uncultured_bacterium;
79.47 1554 166 122 9 1464 1 1499 0.0 961
Query3 [0]Bacteria; [1]Proteobacteria; [2]Deltaproteobacteria; [5]uncultured_delta_proteobacterium; [6]uncultured_delta_proteobacterium;
82.33 928 124 40 446 1352 1151023 0.0 769
Query4 [0]Bacteria; [1]Proteobacteria; [2]Gammaproteobacteria; [3]Pseudomonadales; [4]Pseudomonadaceae; [5]Pseudomonas; [6]Pseudomonas_sp.;
81.97 976 112 61 428 1371 401 1344 0.0 769

3. Results and Discussion

In this section, we present experiments that evaluate our algorithms performance. In the case of MPI-blastn, the performance results were compared with mpiBLAST[20] and ScalaBLAST,[21] which represent state-of-the-art parallel BLAST implementations for clusters. For NCBI-TaxCollector, the results were compared with the original TaxCollector[24] that is, to the best of our knowledge, the only existing tool for taxonomic attachment in BLAST results. We evaluated the algorithms in terms of execution time and speedup. Speedup is a metric widely used in high-performance computing to evaluate how many times faster a parallel code is when compared to a corresponding sequential one.

The experiments were performed on a High Performance Computing (HPC) cluster composed of 16 Dell PowerEdge M610 nodes. Each node has 2 Intel(R) Xeon(R) CPU E5645 2.40 GHz processors, each with 6 cores, capable of running 12 threads simultaneously. These processors support Hyper-Threading (HT) technology, which give us a total of 24 cores per node. The nodes are interconnected by an Infiniband network and message exchanging is carried out by Open MPI library,[29] running on a standard Ubuntu Linux version 10.04. Each node of the cluster has a 24 GB RAM memory, a single local disk and shared storage. The cluster uses a network file system (NFS) to store user data. For mpiBLAST execution, the sequence database was split into as many partitions as the number of worker processes.

3.1. *MPI-blastn performance*

In order to evaluate MPI-blastn performance under different workloads, we used two sizes of input data: 30,000 and 100,000 lines. These input sequences were randomly collected from NCBI nt database.[6] The parallel BLAST search tests were performed over the original nt database. Three trials were run for each case, showing a standard deviation of less than 0.001 in each.

Our first experiment was a direct comparison of MPI-blastn and mpiBLAST in our test cluster (Fig. 3) against the sequential time of BLAST+ running in one core. Using the input of 30,000 lines (average sequence lengh of 1074.08 nucleotides, standard deviation of 944.16 nucleotides), we compared the two implementations scalability up to 240 cores (in our case 10 nodes). The output files were generated in tabular format for all experiments. Although our test cluster has 16 nodes (384 cores), we were unable to execute mpiBLAST beyond 244 cores (the program indicates in an error message that is not able to go beyond this number of cores).

Our first measurements show that our implementation scales in general better than mpiBLAST even for this small workload. Soon after, 24 cores mpiBLAST speedup is already stagnating and begins to decline after 96 cores. MPI-blast, on the other hand, keeps scaling linearly compared to the theoretical speedup up to 240 cores for this workload. We believe this huge difference in performance is caused mainly by the BLAST version used in the implementation and in the more optimized parallelization strategy. MPI-blastn is based on a recent version of BLAST (2.2.25)

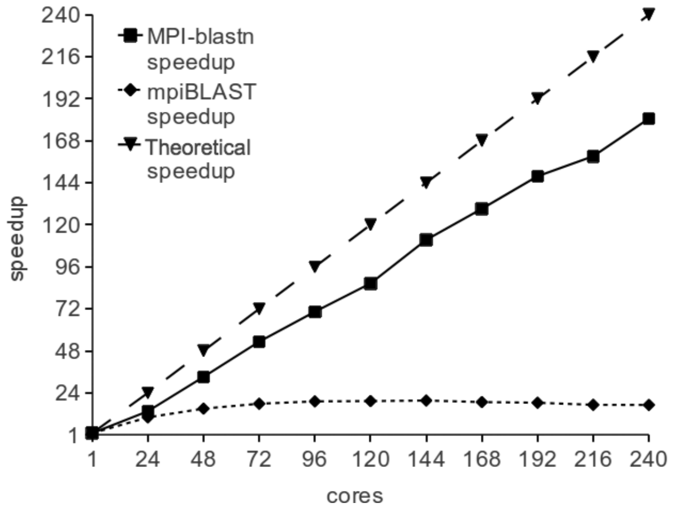


Fig. 3. Performance comparison between MPI-blastn and mpiBLAST using 30,000 input lines in a cluster with 10 nodes (240 cores).

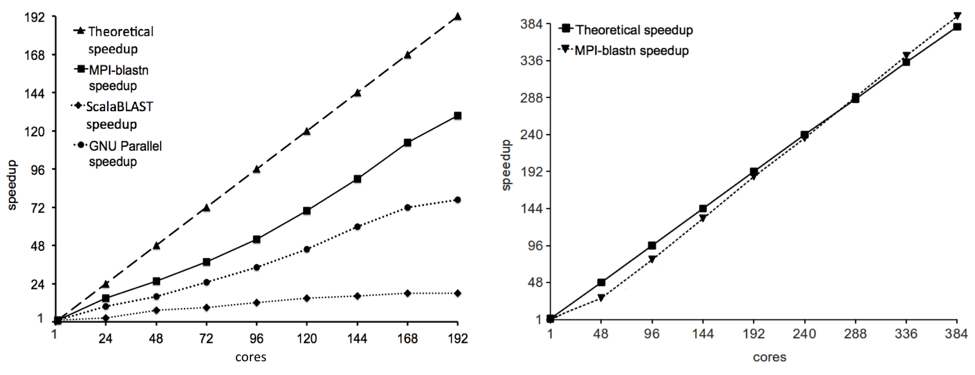
called BLAST+, since mpiBLAST is based on a previous version of BLAST (2.0.9). BLAST+ shows a more efficient algorithm with substantial improvements than the previous versions.[16] Furthermore, choosing a simpler approach to partition the problem, subdividing only the queries and giving full copies of the NCBI database to the slave processes, our implementation was able to decrease the execution time of BLAST+ significantly, from 7.5 hours to less than 3 min (Table 2). This result represents an execution up to 186 times faster for 240 computational cores, if compared to the sequential time, which represents 95.83% of the expected performance (theoretical speedup) for this workload (30,000 lines). Table 2 also shows that extending our tests to all the 16 nodes of the cluster (384 cores) and with a bigger workload (100,000 lines, average sequence lengh of 1110.19 nucleotides, standard deviation of 773.41 nucleotides), we obtained an even better performance, reducing the execution time from 37 hours to 5.5 min.

We also evaluate our implementation scalability up to 16 cluster nodes (384 cores) for a medium workload (100,000 input lines). We see that this heavier workload improved the cores efficiency due to a better distribution of task sizes, decreasing the idle time among processes and increasing the performance (Fig. 4). As a result, MPI-blastn performance obtained is very close to the theoretical speedup, even surpassing it slightly when executing over 288 cores. This superlinear behavior may be the result of cache effects, when task sizes are small enough to fit entirely in the cache and profit from much lower memory latency. The results also indicate that MPI-blastn scalability was not affected by the standard deviation of the input sequence length (773.41 nucleotides). The effects of sequence length on scalability of our implementation may have been minimized because MPI-blastn distributes the workload evenly

Table 2. Execution time (hours) for mpiBLAST and MPI-blastn with different input sizes in a cluster of 16 nodes (384 cores).

Input size (lines)	30.000	30.000	100.000
Number of cores	mpiBLAST	MPI-blastn	MPI-blastn
1	7.43	7.43	36.80
4	3.54	1.77	10.20
8	2.05	0.98	5.81
12	1.52	0.88	4.27
16	1.32	0.79	3.72
20	1.19	0.63	3.15
24	0.74	0.55	2.70
48	0.49	0.22	1.34
72	0.42	0.14	0.71
96	0.39	0.11	0.48
120	0.38	0.09	0.36
144	0.38	0.07	0.28
168	0.40	0.06	0.23
192	0.40	0.05	0.20
216	0.43	0.05	0.18
240	0.43	0.04	0.16
264	X	—	0.14
288	X	—	0.13
312	X	—	0.12
336	X	—	0.11
360	X	—	0.10
384	X	—	0.09

Note: X: execution for this number of cores is not supported by mpiBLAST.



(a) (b)

Fig. 4. Scalability of MPI-blastn: (a) Comparison among MPI-blastn, GNU Parallel and ScalaBLAST using 30,000 input lines in a cluster with 8 nodes (192 cores) and (b) MPI-blastn scalability using 100,000 input lines in a cluster with 16 nodes (384 cores).

among all processes. However, further studies are needed for understanding the influence of sequence length on BLAST performance and MPI-blastn scalability.

We also evaluated ScalaBLAST as another parallel solution to be compared with MPI-Blastn. Due to the fact that the ScalaBLAST requires a ratio between the number of cores and the amount of RAM per core (at least 8 GB RAM for our testing environment), the evaluations between ScalaBLAST and MPI-blastn were performed on another HPC cluster using eight machines with 3.3 GHz 24-core Opteron 6,378 processor and 250 GB RAM per node. The nodes are interconnected by either Gigabit Ethernet or Infiniband network. For ScalaBLAST evaluation, we used the following parameters: Disk group size and cores per node = number of cores per node; task group size = total number of cores −1; and first sub manager = 1.

Table 3 shows the performance evaluation of ScalaBLAST. We configured ScalaBLAST for allocating 8 GB and 10 GB of RAM per core because of its RAM ratio requirement. Figure 4 shows the scalability results for the best performance settings of ScalaBLAST. Based on the results presented, MPI-blastn has shown two main advantages: It has less hardware requirements such as minimum memory per core (<1 GB RAM per node), and it has presented a shorter execution time than the ScalaBLAST. ScalaBLAST keeps separate copies of the database and query list in RAM on each core and the rest of the RAM is reserved for BLASTs compute kernel, which can be memory consuming. If this amount of separate information needs to be exchanged through nodes in a cluster, this may also cause network overload. This limitation did not allow ScalaBLAST scalling when using less than 8 GB RAM per

Table 3. Execution time (hours) for ScalaBLAST, GNU Parallel, and MPI-blastn with input size of 30.000 lines in a cluster of 8 nodes (192 cores).

RAM allocated per core	8 GB	1 GB	10 GB	1 GB	1 G
Network	Gigabit Ethernet	Gigabit Ethernet	Infiniband	Infiniband	Infiniband
Number of cores	ScalaBLAST	MPI-blastn	ScalaBLAST	MPI-blastn	GNU Parallel
1	8.19*	8.19*	1.80*	1.80*	1.80*
4	X	—	X	0.53	0.50
8	X	—	X	0.26	0.25
12	X	—	1.25	0.18	0.23
16	X	—	0.99	0.16	0.21
20	X	—	0.83	0.13	0.20
24	X	—	0.73	0.12	0.19
48	X	—	0.25	0.07	0.11
72	7.17	0.15	0.20	0.05	0.07
96	0.26	0.13	0.15	0.04	0.05
120	0.19	0.11	0.12	0.03	0.04
144	0.18	0.06	0.11	0.02	0.03
168	0.17	0.04	0.10	0.02	0.03
192	0.13	0.01	0.10	0.01	0.02

Note: X: This number of cores is not supported due to RAM per core ratio requirements.
*Execution times refer to sequential BLAST version used as base for ScalaBLAST implementation.

core. We suppose that the RAM requirement of ScalaBLAST and the way that it manages the database and query sequences separately in the RAM may have caused this performance difference in the present HPC environment. Nevertheless, ScalaBLAST may improve its performance in HPC clusters that have higher ratio of RAM available per core and fast network such as Infiniband (Table 3).

Beyond performance, we observed some advantages in MPI-blastn workflow when compared to ScalaBLAST. For example, the user does not have to deal with preconfiguration operations for running the MPI-blastn. In ScalaBLAST, however, the user must configure a parameter file (sb_param.in) that defines workflow rules, task group sizes, task distribution methods, sizing of memory usage, etc.

We also evaluated GNU parallel performance compared to MPI-blastn. The results indicate that our implementation scales in general better than GNU Parallel. MPI-blastn speedup shows an average improvement of more than 30% if compared to GNU Parallel speedup (Fig. 4). Unlike MPI-blastn, GNU parallel uses multiple SSH connections. Cryptographic methods are central to SSH and these methods may affect performance due to encryption overhead, whereas MPI is optimized for message exchange and does not have encryption. In addition, MPI is executed in the transport layer, whereas SSH runs in the application layer. This may increase the packet size, causing more overhead if compared to MPI.

3.2. *NCBI-TaxCollector performance*

In order to evaluate our algorithm performance, we compared its performance with a similar tool. Currently, a recent version of TaxCollector by Giongo *et al.*[24] is the only program available to perform the taxonomic attachment task. The input files used in our performance evaluation consist of the MPI-blastn results in tabular text format. The MPI-blastn results were generated from the sequences and databases described in previous section. TaxCollector takes about 5 min of execution time for 4 lines of MPI-blastn results, and needs more than 60 GB of RAM to perform the task (about 75 s per input line of BLAST results). Following this execution example, a taxonomic attachment that takes 2 days with TaxCollector, now can be performed in less than 1 hour with NCBI-TaxCollector. However, we could not extend our tests with TaxCollector for larger input sizes due to the large amount of memory usage necessary in order to run TaxCollector. Unlike this tool, the present implementation of NCBI-TaxCollector takes less than 1 s to run the same input. Furthermore, we observed a max RAM memory usage of less than 0.5 GB (lower than the 60 GB used by TaxCollector). This result shows that NCBI-TaxCollector is able to perform taxonomic attachments 125 times faster and needs 120 times less RAM than TaxCollector by Giongo *et al.*[24]

Extending the evaluation of the present tool, we increased the input size (number of MPI-blastn results) up to 450,000 lines. We performed this experiment in order to assess the behavior of our algorithm for larger input sizes, in matters of execution time and memory usage. Varying the input sizes, we see that the execution time and

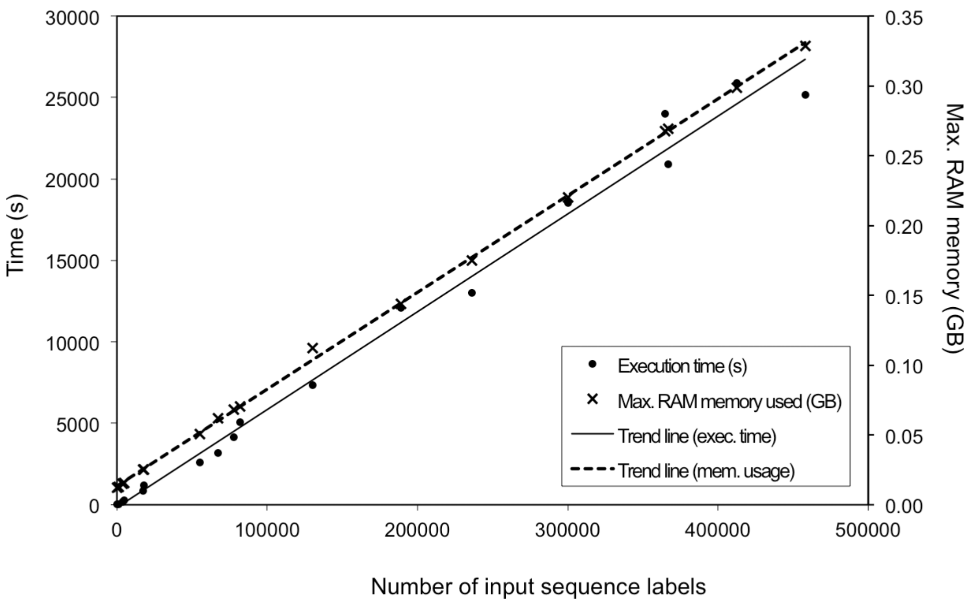


Fig. 5. A NCBI-TaxCollector performance: Execution time and memory usage trends versus input size.

memory usage remain a linear proportion if compared with the input size. This result is demonstrated in Fig. 5. The same evaluation was impossible to be performed with TaxCollector due to the large amount of memory usage and unfeasible execution time.

4. Conclusions

Sequence search and post processing of these results has become subject of many high performance optimizations. The exponential increase in the size of data generated by next generation sequencing technologies has made currently used tools impossible to be executed in a feasible time, unless optimizations of parallelism and lower complexity are performed on their algorithms. In this work, we exemplify this issue with BLAST and TaxCollector programs. Without optimizations, these tools may take weeks, even months, to perform a metagenomic analysis with the current size of data generated. In this paper, we present a new parallel implementation based on a recent version of BLAST+ (2.2.25) at NCBI.[16] Unlike BLAST+, that can only take advantage of individual shared-memory multicore machines, our implementation exploits the computing power of clusters of multicore machines, to allow increased performance and scalability. The resulting tool, called MPI-blastn obtained a super linear speedup if compared to the theoretical value, running 408 times faster in 384 cores. Furthermore, if compared with other parallel BLAST tools, our implementation shows a much higher performance than mpiBLAST[20] and ScalaBLAST.[21]

We also present our new NCBI-TaxCollector implementation. The algorithm converts the taxonomic databases from ASCII to binary format. This conversion remarkably speeds up the search, decreases the algorithm complexity, and also reduces the total size of NCBI database files. Besides, comparing the NCBI-Tax-Collector execution with an adapted version of TaxCollector,[24] we see not only a remarkable decrease of runtime but also in RAM usage. Our evaluations demonstrated that NCBI-TaxCollector is able to perform taxonomic attachments 125 times faster and needs 120 times less RAM memory than adapted version of TaxCollector by *et al.*[24]

Combining both tools, we offer a real improvement for metagenomic analysis. For example, a multiple sequence search that currently takes one week can be performed in 25 min and a post processing with NCBI taxonomic data attachment, which takes 48 hours, now is able to run in less than half hour. In a future work, we intend to expand our optimization of nucleotide sequence search to protein sequence search, in order to improve the performance of protein function prediction for environmental genomics data.[30] For NCBI-TaxCollector, a further improvement to use multiprocessor and/or multicomputer parallelism is possible to be implemented, since the taxonomic attachment for one input line is independent from another. For MPI-blastn implementation, due to variations in the size of sequences and consequently in the workload size by process, new job scheduling strategies may be implemented to handle such problems.

References

1. Proctor GN, Mathematics of microbial plasmid instability and subsequent differential growth of plasmid-free and plasmid-containing cells, relevant to the analysis of experimental colony number data, *Plasmid* **32**(2):101–130, 1994.
2. Kaeberlein T, Lewis K, Epstein SS, Isolating "uncultivable" microorganisms in pure culture in a simulated natural environment, *Science* **296**(5570):1127–1129, 2002.
3. Handelsman J, Tiedje J, Alvarez-Cohen L, Ashburner M, Cann I, Delong E, Doolittle W, Fraser-Liggett C, Godzik A, Gordon J, The new science of metagenomics: Revealing the secrets of our microbial planet, *Nat Res Council Report* **13**, 2013.
4. Kahn SD, On the future of genomic data, *Science* **331**(6018):728–729, 2011.
5. HiSeq Systems — Illumina. Available at http://www.illumina.com/systems/hiseq_systems.ilmn (accessed 10 Jan 2013).
6. Geer LY, Marchler-Bauer A, Geer RC, Han L, He J, He S, Liu C, Shi W, Bryant SH, The NCBI BioSystems database, *Nucleic Acids Res* **38**(Database issue):D492–D496, 2010.
7. DeSantis TZ, Hugenholtz P, Larsen N, Rojas M, Brodie EL, Keller K, Huber T, Dalevi D, Hu P, Andersen GL, Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB, *Appl Environ Microbiol* **72**(7):5069–5072, 2006.
8. Cole JR, Chai B, Farris RJ, Wang Q, Kulam-Syed-Mohideen AS, McGarrell DM, Bandela AM, Cardenas E, Garrity GM, Tiedje JM, The ribosomal database project (RDP-II): Introducing myRDP space and quality controlled public data, *Nucleic Acids Res* **35** (Database issue):D169–D172, 2007.
9. Giongo A, Crabb DB, Davis-Richardson AG *et al.*, PANGEA: Pipeline for analysis of next generation amplicons, *ISME J* **4**(7):852–861, 2010.

10. Schloss PD, Westcott SL, Ryabin T *et al.*, Introducing mothur: Open-source, platform-independent, community-supported software for describing and comparing microbial communities, *Appl Environ Microbiol* **75**(23):7537–7541, 2009.
11. Cole JR, Chai B, Farris RJ, Wang Q, Kulam SA, McGarrell DM, Garrity GM, Tiedje JM, The Ribosomal Database Project (RDP-II): Sequences and tools for high-throughput rRNA analysis, *Nucleic Acids Res* **33**(Database issue):D294–D296, 2005.
12. Healy MD, Using BLAST for performing sequence alignment, *Current Protocols Human Genetics*, Chapter 6:Unit 6.8, Wiley Online Library, 2007.
13. de Araujo Macedo E, Magalhaes Alves de Melo AC, Pfitscher GH, Boukerche A, Hybrid MPI/OpenMP strategy for biological multiple sequence alignment with DIALIGN-TX in heterogeneous multicore clusters, *Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW), 2011 IEEE Int Symp*, pp. 418–425, IEEE, 2011.
14. Vouzis PD, Sahinidis NV, GPU-BLAST: Using graphics processors to accelerate protein sequence alignment, *Bioinformatics* **27**(2):182–188, 2011.
15. Rezaei S, Monwar MM, Bai J, Performance comparison of MPI-based parallel multiple sequence alignment algorithm using single and multiple guide trees, *Cognitive Informatics, 5th IEEE Int Conf*, pp. 595–600, 2006.
16. Tringe SG, Rubin EM, Metagenomics: DNA sequencing of environmental samples, *Nat Rev Genet* **6**(11):805–814, 2005.
17. Kurokawa K, Itoh T, Kuwahara T *et al.*, Comparative metagenomics revealed commonly enriched gene sets in human gut microbiomes, *DNA Res* **14**(4):169–181, 2007.
18. Maidak BL, Cole JR, Lilburn TG, Parker CT, Jr., Saxman PR, Farris RJ, Garrity GM, Olsen GJ, Schmidt TM, Tiedje JM, The RDP-II (Ribosomal Database Project), *Nucleic Acids Res* **29**(1):173–174, 2001.
19. Camacho C, Coulouris G, Avagyan V, Ma N, Papadopoulos J, Bealer K, Madden TL, BLAST+: Architecture and applications, *BMC Bioinformatics* **10**:421, 2009.
20. Lin HS, Ma XS, Feng WC, Samatova NF, Coordinating computation and I/O in massively parallel sequence search, *IEEE T Parall Distr* **22**(4):529–543, 2011.
21. Oehmen C, Nieplocha J, ScalaBLAST: A scalable implementation of BLAST for high-performance data-intensive bioinformatics analysis, *IEEE T Parall Distr* **17**(8):740–749, 2006.
22. Tange O, GNU parallel — the command-line power tool, *The USENIX Magazine* **36**(1):42–47, 2011.
23. Rapier C, Bennett B, High speed bulk data transfer using the SSH protocol, *Proc 15th ACM Mardi Gras Conf*, pp. 11, 2008.
24. Giongo A, Davis-Richardson AG, Crabb DB, Triplett EW, TaxCollector: Modifying current 16S rRNA databases for the rapid classification at six taxonomic levels, *Diversity* **2**(7):1015–1025, 2010.
25. NCBI Taxonomy database. Available at ftp://ftp.ncbi.nih.gov/pub/taxonomy/.
26. Benson DA, Karsch-Mizrachi I, Clark K, Lipman DJ, Ostell J, Sayers EW, GenBank, *Nucleic Acids Res*, 2011.
27. Sait SM, Al-Mulhem M, Al-Shaikh R, Evaluating BLAST runtime using NAS-based high performance clusters, *Computational Intelligence, Modelling and Simulation (CIMSiM), 2011 Third Int Conf*, pp. 51–56, 2011.
28. Black PE, Big-O notation, *Dictionary of Algorithms and Data Structures*, 2007.
29. Graham RL, Woodall TS, Squyres JM, Open MPI: A flexible high performance MPI, *Lect Notes Comput Sci* **3911**:228–239, 2006.
30. Wong L, Introduction — some new results and tools for protein function prediction, RNA target site prediction, genotype calling, environmental genomics, and more, *J Bioinform Comput Biol* **9**(6):v–vii, 2011.

Raquel Dias received BS Degree in Biology from the Catholic University of Rio Grande do Sul — (PUCRS, Porto Alegre, RS, Brazil, 2010) and MSc degree in Computer Science from the PUCRS, 2012. Since 2012, she is a PhD student at the Microbiology and Cell Science Department of the Institute of Food and Agricultural Sciences at the University of Florida. Her research interests include phylogenetic reconstruction, high-throughput DNA sequencing analysis, genome assembly, ancestral sequence reconstruction, and structural biochemistry.

Miguel G. Xavier holds a degree in Information Systems from the Pontifical Catholic University of Rio Grande do Sul — (PUCRS, Porto Alegre, RS, Brazil, 2011), MSc in Computer Science from the Pontifical Catholic University of Rio Grande do Sul — (PUCRS, Porto Alegre, RS, Brazil, 2013), and MBA in Project Management Software from the Pontifical Catholic University of Rio Grande do Sul — (PUCRS, Porto Alegre, RS, Brazil, 2012). He is currently a researcher at the PUCRS and a consultant in Software Development in a variety of companies. He has vast experience in the area of Computer Science, Specializing in the following topics: Cloud Computing, Virtualization, Parallel and Distributed Processing, and Networks.

Fábio Diniz Rossi received his Master's degree in Computer Science from the Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil (2008), and Bachelor's degree from the University of the Region of Campanha, Brazil (2000). He is currently pursuing PhD in Computer Science at the Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil. His primary research interests are cloud computing and energy efficiency.

Marcelo Veiga Neves received his Master degree in Computer Science from the Federal University of Rio Grande do Sul, Brazil (2009), and Bachelor's degree from the Federal University of Santa Maria, Brazil (2005). He is currently pursuing PhD in Computer Science at the Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil. His primary research interests are high performance computing, big data, and computer networks.

Timoteo Lange holds a degree in Technology of Data Processing from the University of Ivaiporã — (Univale, Ivaiporã, PR, Brazil, 1999), specialist degree in Network Computer from Federal Institute of Paraná (IFPR, Curitiba, PR, Brazil, 2002), and Master Degree in Computer Science from the Catholic University of Rio Grande do Sul — (PUCRS, Porto Alegre, RS, Brazil, 2013). He is currently a professor in computer Network and operating systems in the Unisc (University of

Santa Cruz do Sul) and works in Procergs (Company of Data Processing of State of Rio Grande do Sul) as Database Administrator in e-government projects.

Adriana Giongo holds BS degree in Biological Sciences from the Federal University of Rio Grande do Sul (UFRGS, Porto Alegre, RS, Brazil, 2000), MSc in Agriculture and Environmental Microbiology from the Federal University of Rio Grande do Sul (PPGMAA-UFRGS, Porto Alegre, RS, Brazil, 2003), PhD in Genetics and Molecular Biology from the same university (PPGGBM-UFRGS, Porto Alegre, RS, Brazil, 2007), and a post-doctoral degree in Microbial Ecology from the University of Florida (UF, Gainesville, FL, USA) from 2008 to 2011. Since 2011, she is a researcher at the CEPAC — Center of Excellence in Research and Innovation in Petroleum, Mineral Resources, and Carbon Storage from Catholic University of Rio Grande do Sul (PUCRS, Porto Alegre, RS, Brazil), working on metagenomics approach applied to the microbial diversity studies.

César De Rose holds BS degree in Computer Science from the Catholic University of Rio Grande do Sul — (PUCRS, Porto Alegre, RS, Brazil, 1990), MSc in Computer Science from the Federal University of Rio Grande do Sul — (CPGCC-UFRGS, Porto Alegre, RS, Brazil, 1993), and a doctoral degree from Karlrsruhe University (Karlsruhe, Germany, 1998). Since 1998, he is a professor at PUCRS and a member of the Parallel and Distributed Processing Group. His research interests include resource management in parallel and distributed architectures and operating systems. Since 2008, he is the lead researcher at the PUCRS High Performance Laboratory (LAD-PCURS).

Eric W. Triplett received BS degree from Rutgers University in Biology, MS degree in Botany from the University of Maryland, and a PhD from the University of Missouri, Columbia in Agronomy/Plant Physiology in 1981. Over time, his interest in bacterial genetics and genomics has grown particularly in plant-microbe interactions. He is now Professor and Chair of the Microbiology and Cell Science Department of the Institute of Food and Agricultural Sciences at the University of Florida. His research interests include high-throughput DNA sequencing analysis, a microbial role in type 1 diabetes, soil microbial ecology, and citrus greening disease.