JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 25, Number 8, 2018

Pp. 871–881
DOI: 10.1089/cmb.2018.0079

Massively Parallel Implementation of Sequence Alignment with Basic Local Alignment Search Tool Using Parallel Computing in Java Library

MAREK NOWICKI[1], DAVIT BZHALAVA[2], and PIOTR BAŁA[3]

ABSTRACT

Basic Local Alignment Search Tool (BLAST) is an essential algorithm that researchers use for sequence alignment analysis. The National Center for Biotechnology Information (NCBI)-BLAST application is the most popular implementation of the BLAST algorithm. It can run on a single multithreading node. However, the volume of nucleotide and protein data is fast growing, making single node insufficient. It is more and more important to develop high-performance computing solutions, which could help researchers to analyze genetic data in a fast and scalable way. This article presents execution of the BLAST algorithm on high performance computing (HPC) clusters and supercomputers in a massively parallel manner using thousands of processors. The Parallel Computing in Java (PCJ) library has been used to implement the optimal splitting up of the input queries, the work distribution, and search management. It is used with the nonmodified NCBI-BLAST package, which is an additional advantage for the users. The result application—PCJ-BLAST—is responsible for reading sequence for comparison, splitting it up and starting multiple NCBI-BLAST executables. Since I/O performance could limit sequence analysis performance, the article contains an investigation of this problem. The obtained results show that using Java and PCJ library it is possible to perform sequence analysis using hundreds of nodes in parallel. We have achieved excellent performance and efficiency and we have significantly reduced the time required for sequence analysis. Our work also proved that PCJ library could be used as an effective tool for fast development of the scalable applications.

Keywords: BLAST, Java, next-generation sequencing, PCJ, sequence alignment.

1. INTRODUCTION

CONTINUOUS GROWTH IN THE SIZE OF THE DATABASES storing nucleotide and protein data is observed with the development of next-generation sequencing (NGS). At the same time there is high demand to extract useful information from these massive data sources. The development of high-performance computing power helps researchers analyze voluminous genetic data. Different hardware such as clusters,

[1]Faculty of Mathematics and Computer Science, Nicolaus Copernicus University in Toruń, Poland.
[2]Department of Laboratory Medicine, Karolinska Institutet, Stockholm, Sweden.
[3]Interdisciplinary Center for Mathematical and Computational Modeling, University of Warsaw, Warsaw, Poland.

supercomputers, or even custom systems can be used to solve the numerous problems originating from analysis of biological and biomolecular data.

In particular, NGS is used for extensive sequencing of the DNA in a sample, bypassing the necessity of cloning or prior knowledge of the analyzed sequence. To understand the taxonomic composition of metagenomic datasets from NGS technologies first raw sequences are assembled into contiguous sequences (contigs) using de novo assembly (de Bruijn graph) algorithms. Assembled contigs are then subjected to taxonomic classification performed by comparing them against available genetic data. The contigs are classified in many ways for example as previously known sequences, a sequence related to previously known ones, or as unrelated to any sequence stored in the databases.

An extensively used bioinformatics application for sequence alignment analysis is an heuristic algorithm called Basic Local Alignment Search Tool (BLAST) (Altschul et al., 1990, 1997). It allows for searching a given query sequence in DNA (nucleotide) and protein sequence databases. As a result, the most similar sequences from listed databases are identified.

BLAST is available on the web as a standalone application to install on local resources. The implementation from the National Center for Biotechnology Information (NCBI) (www.ncbi.nlm.nih.gov/BLAST; accessed November 6, 2017) is the most popular one and considered as a reference.

NCBI-BLAST can be run in multithreading environment like in symmetric multiprocessing systems, however, only on one single node. NCBI-BLAST keeps a regular linear speed-up behavior, but node size limits the overall performance significantly.

Our proposed approach uses the nonmodified NCBI-BLAST application to be sure that usage of further releases of the software will require no additional work. Our solution is based on input sequence fragmentation and efficient query partitioning ensuring load balancing. The main advantage of this approach is the possibility of using hundreds and thousands of processor cores that can significantly reduce the time necessary to perform an analysis. Moreover, the user gets the same results as running the original NCBI-BLAST.

This article is an extended version of the proceedings article (Nowicki et al., 2017) that contains a preliminary version of our research on usage of Parallel Computing in Java (PCJ) library to parallelize sequence alignment with BLAST. The current version was extensively extended and enhanced. The main improvements are as follows: the detailed performance data used to evaluate Parallel Computing in Java (PCJ)-BLAST is provided, the evaluation of multithreaded NCBI-BLAST execution with full node occupation is presented, and performance results of the parallel I/O are given. The performance has been compared with other solutions.

Since BLAST execution takes much time and requires significant computational resources, there are known attempts to run BLAST in a multinode manner, on clusters and supercomputers, so there is need to parallelize the execution. We should acknowledge rising effort to run BLAST on the cloud (Celesti et al., 2017), however, in this article we focus on HPC type of systems.

The bioinformatics community is still lacking good and efficient tools that would fulfill their needs and for utilizing available computer architectures aligned with the recent progress of NCBI software, even in spite of numerous efforts in the area of parallelization of BLAST execution.

There are mainly two approaches to make BLAST execute on multinode systems: first is based on the distribution of the query set across computing nodes (Chi et al., 1997; Braun et al., 2001; Cofer, 2012), second is done by partitioning the database among computing nodes (Bjornson et al,. 2002; Mathog, 2003). In the second approach, one search is performed at a time, but on a smaller portion of the database on each node, which is the main advantage, as it reduces the database size, each node has to load into memory and looked at. This solution was first implemented in the mpiBLAST (Darling et al., 2003), was further developed in pioBLAST (Lin et al., 2005), and is basis for modest supercomputer implementations (Seetharam et al., 2015; Mikailov et al., 2017).

Therefore, we have developed our solution, which is hardware independent and adjusted to widely used HPC architectures. The compliance to the NCBI-BLAST is a key functionality provided.

The article has the following organizations: Section 2 presents the PCJ library for parallelization Java applications. The following section presents an implementation of the sequence alignment using the PCJ library. Section 4 presents performance and scalability benchmarks. Finally, conclusions and plans are highlighted.

2. PCJ LIBRARY

PCJ (http://pcj.icm.edu.pl; accessed November 6, 2017) is a Java library (Nowicki et al., 2014) that allows performing parallel computations on a single workstation and distributed systems. It can work on the

multicore systems with the various interconnects, such as Ethernet or InfiniBand. PCJ source code is available on GitHub under Open Source license (BSD).

PCJ implements Partitioned Global Address Space (PGAS) model. The library development was inspired by the lack of such solution for the recent Java releases and by other solutions, such as Co-Array Fortran (Numrich and Reid, 1998), Unified Parallel C (Carlson et al., 1999), or Titanium (Hilfinger et al., 2005). PCJ library provides users with the uniform view across nodes. In PCJ design, we ensured compliance with Java standards, and additional libraries or tools, which are not part of the Java distribution, are not required.

In PCJ, as it is presented in Figure 1, each PCJ thread (task) executes its own set of instructions and has its local memory. By default, instructions and variables are local to the task, but some of them can be marked as shareable and can be accessed from other PCJ threads. The library provides methods to perform basic operations such as synchronization of PCJ threads and data transfer between them. The data transfer operations are executed in a one-sided asynchronous way. With PCJ library, the user can also create tasks' groups, broadcast data within a group, and monitor variable updates.

The application using PCJ library is executed as ordinary Java application. In the single-node execution, it uses Java Virtual Machine (JVM). While running in the multinode setup, one JVM is started on each node. One JVM instance is a PCJ node. Single PCJ node can hold many PCJ threads. Such design is well suited for current computer architectures containing a large number (hundreds or thousands) of nodes, each of them built on several or even more cores.

3. PCJ-BLAST

PCJ-BLAST is a sequence alignment application implemented in Java with the help of the PCJ library. The application allows running multiple NCBI-BLAST instances in parallel on multinode systems with the possibility of postprocessing generated output. The parallelization is based on the distribution of query sequences among used nodes. The query sequences are searched against a reference database. This approach seems to be well adjusted to the NGS data that contains a large number of short sequences searched independently against the nucleotide sequence database. The reference database from NCBI repository contains a total of 20,485,291 sequences as of December 2015 and has a size of 52 GB.

3.1. Input data

BLAST input file is obtained from the NGS sequencing equipment (i.e., Illumina Sequencing). The file consists of about 1 million short, 100–150 character-long, sequences (*reads*) in FASTA format: each

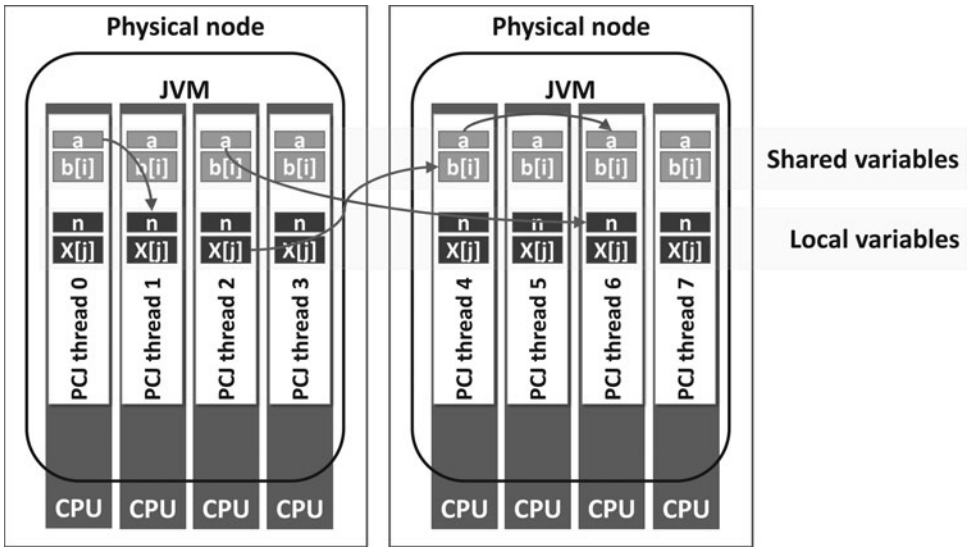

FIG. 1. Diagram of PCJ computing model [from Nowicki et al. (2016) p. 67]. Arrows present possible communication using put() or get() methods acting on shared variables. PCJ, Parallel Computing in Java.

sequence has a one-line human-readable description, denoted by the greater-than (>) symbol, then one or more nonblank lines containing sequence data. Next sequence is denoted by the description line starting with the greater-than sign.

The processing of each sequence (i.e., search in the reference database) can take a totally different time, ranging from a couple up to thousands of seconds (Fig. 2), despite the fact that the sequences included in the input file are of similar length.

The large size of the input data allows us to speed-up sequence alignment based on the query-parallelized approach. Multiple query searches are distributed among different nodes and can be done independently. Moreover, query parallelization can be implemented through a *wrapper* to the original NCBI-BLAST. The scheduling has to be performed dynamically during execution since the analysis time cannot be estimated based on the input sequence. The decision is performed based on the current state of the processors' workloads.

3.2. Parallel sequence alignment

The PCJ processes sequences in parallel using multiple instances of NCBI-BLAST executed with multiple threads at a node. The number of nodes and NCBI-BLAST threads can be configured during job submission. The FASTA input file is not modified.

The PCJ thread with id zero (*Scatterer*) is used for reading the FASTA input file and distributing sequences. Other PCJ threads (*Workers*) are used for running NCBI-BLAST on sequences and processing BLAST output into a proper format for any further processing (Fig. 3).

The user specifies the number of sequences that are read by the *Scatterer*. Then the *Scatterer* sends sequences to available *Workers* or waits until there is one available. The number of sequences processed by the individual worker at each round is kept low (in our case it is 2) to obtain good load balancing. The *Worker* receives sequences in advance to avoid waiting for data to process. The sequences sent from the *Scatterer* to the *Workers* are not saved in the files. The *Worker* starts BLAST execution with the given sequences. After completing execution, the *Worker* processes the output (XML) into a form suitable for further processing and appends it into its own output file. As the order of sequences is not important for further processing, after processing the input file, outputs are concatenated into one result file. All communication between PCJ threads is asynchronous and does not involve saving data to disk, which minimizes time spent on data exchange between *Scatterer* and *Workers*.

NCBI-BLAST execution time depends significantly on the query parameters, and in our case is as follows:

```
-word_size 11 -gapopen 0 -gapextend 2 -penalty -1-reward 1
-max_target_seqs 10 -evalue 0.001-shows_gis -outfmt 5
```

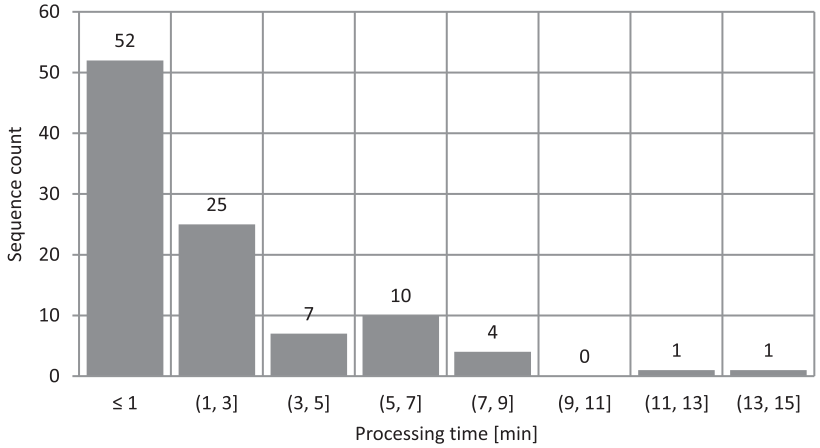

FIG. 2. Histogram of the execution time for the selected reads from the input file. The execution time has been measured using single-threaded execution of the NCBI-BLAST 2.2.28+. NCBI-BLAST, National Center for Biotechnology Information-Basic Local Alignment Search Tool.

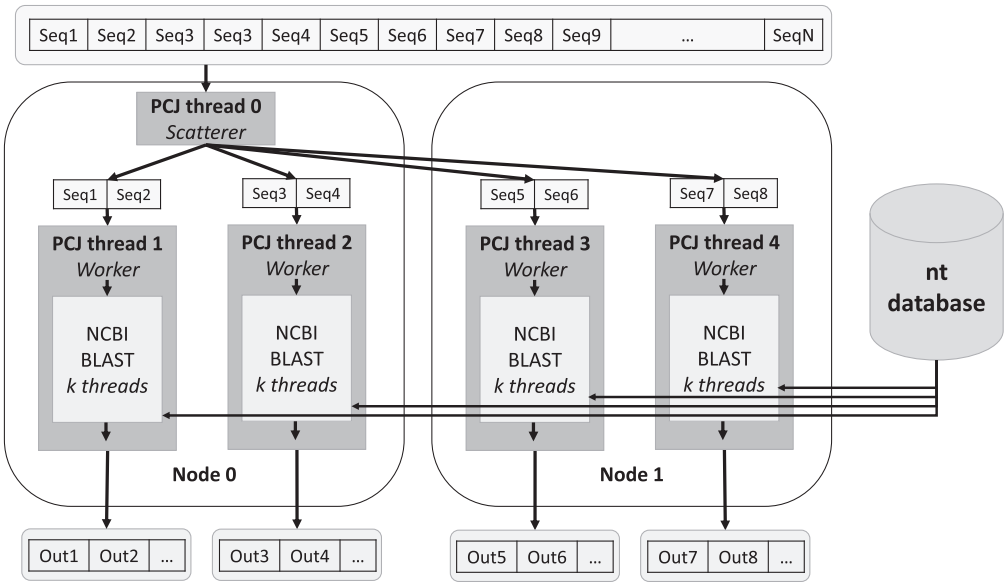

FIG. 3. Schematic view of PCJ-BLAST execution. The sequences are distributed to the available *Workers* based on the dynamic information sent by the *Workers*. The input sequences can be processed in arbitrary order.

These parameters were tuned for the viruses' sequences search, and their optimization was performed out of this work. The output parameter (-outfmt) sets the output to be formatted as an XML file that can be processed by PCJ-BLAST. It is also possible to write output to the file and in that situation, as the outputs of each BLAST execution are concatenated, the resulting file has to be postprocessed to generate a proper XML file. Postprocessing also can be done using PCJ-BLAST.

3.3. Hardware resources

For the performance evaluation, we have used two systems: x86 cluster and Cray XC40, both located at ICM, University of Warsaw.

HPC cluster used for experiments consists of 223 computing nodes equipped with Intel Xeon E5-2697 v3 CPU (28 cores each) with InfiniBand interconnect. Processors are clocked at 2.60 GHz. Every computational node has at least 64 GB of RAM. Nodes are equipped with InfiniBand FDR and 1 Gb Ethernet cards. The HPC cluster has NFS 3.0 and Lustre filesystems. The NFS server used is Hitachi HNAS with HUS150 storage filled with 12k RPM SAS disks. The filesystem is mounted using 10 Gbit ethernet sockets. The Lustre version used on both server and client side is 2.1. The Lustre filesystem has 1 MDS server, 6 OSS servers with 36 OST (DDN SFA12K). For the storage, SSD disks are used (Netapp EF560). Data transfer is provided by Infiniband FDR (56 Gbit/s).

Cray XC40: The system used has 1084 nodes equipped with 2 Intel Xeon E5-2690 v3 CPUs running at 2.60 GHz and 128 GB of RAM. Nodes are connected with Cray's Aries interconnect.

3.4. Multithreaded NCBI-BLAST

The number of NCBI-BLAST instances that can be executed in parallel on each physical node and its thread count can be configured in PCJ-BLAST. Both numbers have to be properly set in accordance with I/O and CPU performance. The number of threads running on a single physical node was the number of PCJ threads multiplied by the number of threads used by each NCBI-BLAST. To use all cores available at the physical node, we kept the number of all threads equal to the number of cores.

To set up optimal configuration, we have run NCBI-BLAST using different numbers of instances and threads. The experiments have been performed using HPC cluster and Cray XC40 systems. In both cases, the input sequence was short (224 and 384 reads long, respectively) and consisted of a duplication of the same reads to ensure ideal load balancing. The input sequence was chosen to provide reasonable workload for the different CPU speed and to minimize computational time used for tests. Processing of single read with 1

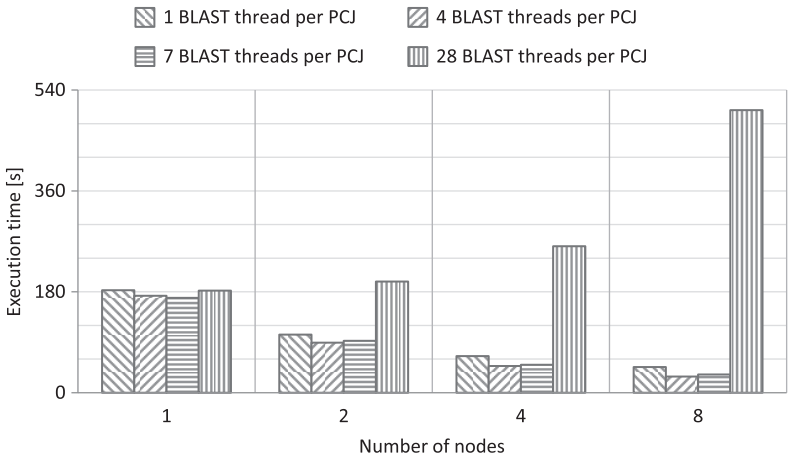

FIG. 4. Execution time for the different number of BLAST instances per node. The execution time has been recorded on the HPC cluster for 224 sequence long input. All sequences in the input file have been identical to avoid variation in the execution time of single sequence.

BLAST thread takes about 30 seconds on Cray XC40. The number of reads was a multiplication (by 8) of the number of processing cores available at each node, which is different for both systems used. The processing of the single read required the same amount of memory. The different CPU speed of both systems does not affect scalability; therefore, differences in performance come mainly from the I/O performance.

3.4.1. HPC cluster. We have performed some performance tests to establish the best scenario for accessing reference database. The results presented in Figure 4 show that for the 28 cores node, optimal configuration is 4 PCJ threads and 7 threads for each NCBI-BLAST instance. The NFS filesystem has been used to read the reference database. There is local disk at each node; however, we decided not to replicate the reference database to minimize file transfer and to simplify software management.

The poor performance in the case when all NCBI-BLAST instances are run, single threaded comes from I/O problems. All instances read the reference database file simultaneously, which causes contention in access to the I/O channel.

3.4.2. Cray XC40. The results presented in Figure 5 show that the performance is similar for the different number of NCBI-BLAST instances run on a single node. However, a bigger number of NCBI-BLAST instances causes more I/O traffic, which influences performance. This effect is much smaller than for HPC cluster. The DVS filesystem (Sugiyama and Wallace, 2008) has been used to read the reference database. There is no disk space on nodes available to store the database locally.

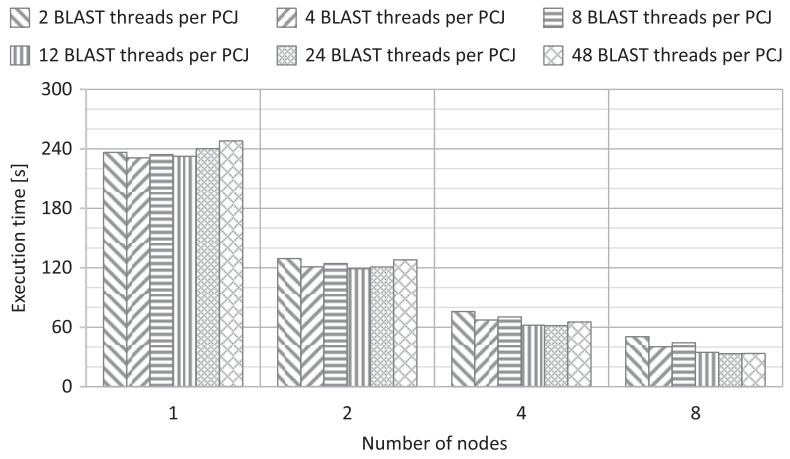

FIG. 5. Execution time for the different number of BLAST instances per node. The execution time has been recorded on the Cray XC40 for 384 sequence (identical) long input.

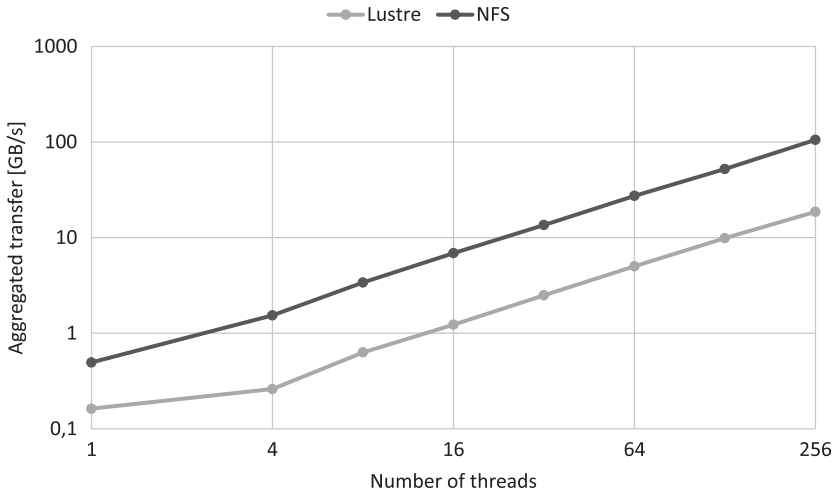

FIG. 6. Comparison of the NFS and Lustre performance at x86 cluster. The aggregated speed of data transfer is presented based on the concurrent read of a single 1 GB input file shared between nodes. Each node is running four PCJ threads reading the file.

3.5. I/O performance

In the query-parallelized approach, the performance depends on the efficient read of the reference database. We have decided to use a single copy of the database for all NCBI-BLAST instances to reduce necessary disk space and to simplify the process of changing the database. Any replication or distribution of the database will make this process more complicated and error prone. Consequently, the database will be read multiple times at each node by each NCBI-BLAST instance. In our case, the reference database is a single file of the size of 52 GB, which is read by each NCBI-BLAST instance. Four PCJ threads were run on each node, reproducing the configuration used for real PCJ-BLAST execution. All threads, from all nodes, were reading data from the same file.

3.5.1. HPC cluster. The overall performance, that is, size of the data read in a second, for the NFS is five to six times higher than for Lustre. The scalability is similar and the difference slightly decreases for larger number of nodes. This result can be explained by the fact that Lustre is designed to serve multiple, large files simultaneously rather than a single file read by multiple threads.

The aggregated speed of data transfer (volume of the data read by all threads in the unit time) is presented in Figure 6.

For PCJ-BLAST, we have observed that processing time of the first set of sequences sent to the workers has taken significantly longer (almost two times) compared with the subsequent runs. This effect has been associated with the synchronous start of all NCBI-BLAST instances. Each of them, upon start, has been reading the reference database, which caused huge I/O traffic and contention in access to the disk. This behavior was not observed in the subsequent runs. The different processing time of the query sequences was different for each NCBI-BLAST instance and access to the disk was asynchronous. To solve the problem of the processing of the first query sequence set, a start of the first NCBI-BLAST instance has been delayed by a random delay up to 100 seconds. This allowed minimizing contention at the beginning of the execution.

3.5.2. Cray XC40. Similar experiments have been performed for the Cray XC40 system. In this case, Lustre filesystem (with read and write access) and Cray DVS (read-only) have been used.

As presented in Figure 7 the DVS performance for a single node is at least two times higher than for Lustre. Scalability is quite similar for both filesystems presenting almost linear scalability up to 256 nodes (1024 threads).

4. PCJ-BLAST PERFORMANCE RESULTS

The experiments were run on the HPC cluster and Cray XC40 systems. In both cases, PCJ-BLAST used PCJ library version 4.1.0. PCJ-BLAST was run using Java JVM v. 1.8.0 from Oracle.

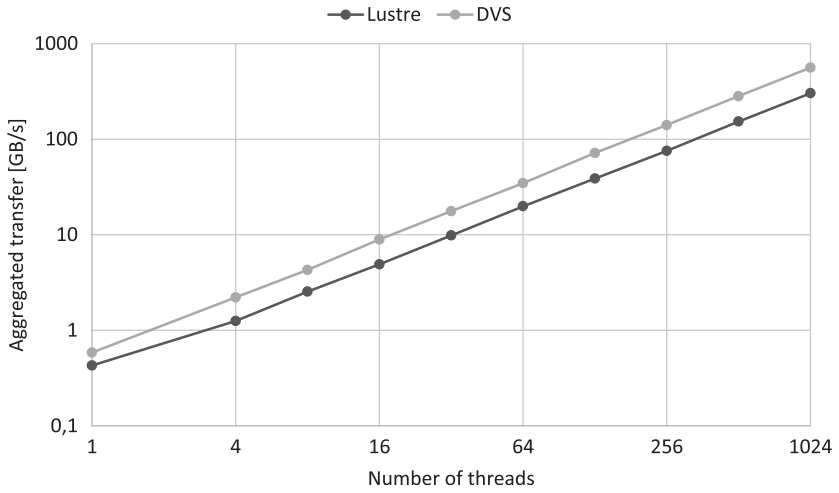

FIG. 7. Comparison of the Cray DVS and Lustre performance on Cray XC40 system. The aggregated speed of data transfer is presented based on the concurrent read of a single 1 GB input file shared between nodes. Each node is running four PCJ threads reading the file.

4.1. HPC cluster

An NFS filesystem mounted at each computing node was used to store the reference database. All NCBI-BLAST copies used this single instance of the reference database. PCJ-BLAST has been run on different number of nodes in the exclusive mode—using all CPU cores available. On each node, PCJ-BLAST executed four NCBI-BLAST instances with seven threads each. The performance data presented in Figure 8 show good scaling up to 64 nodes (1792 cores). The parallel efficiency calculated with respect to 16 nodes (Table 1) is high, close to 0.9, for a large number of used cores. The analysis was made on a query file that contained 1,066,288 sequences.

The analysis time has been reduced from estimated 52 days when using single 28-core node to the slightly more than 1 day (26.5 hours) while using 64 nodes.

4.2. Cray XC40

The read-only DVS filesystem was mounted on each computing node and was used for storing the reference database. PCJ-BLAST was run on different number of nodes under similar conditions as for the HPC cluster—on each node, 4 NCBI-BLAST instances with 12 threads each (to fill the whole available cores) were executed. On each node, hyperthreading (HT) was used gaining almost double speedup

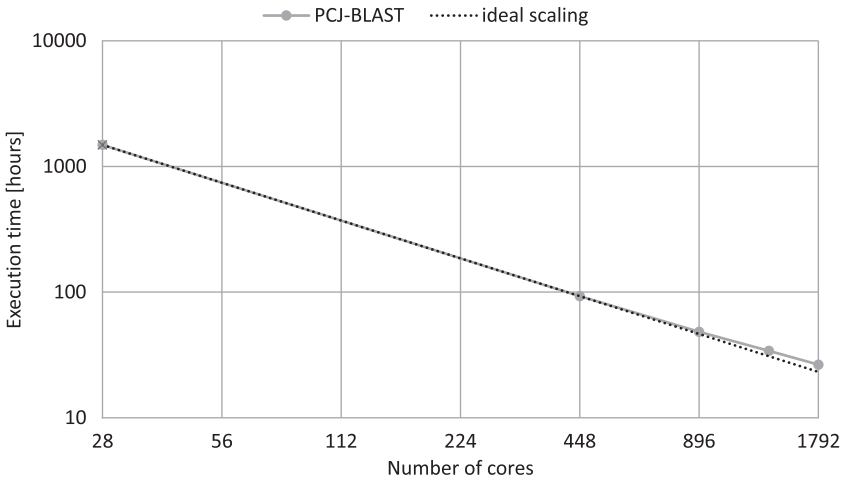

FIG. 8. Performance of PCJ-BLAST on the HPC cluster. The ideal scaling is plotted for reference. The scaling is based on PCJ-BLAST execution on 16 nodes, 4 NCBI-BLAST instances with 7 threads each.

Table 1. Parallel Efficiency of PCJ-BLAST Running on HPC Cluster

		PCJ-BLAST (this work)	
		Four NCBI-BLAST per node	
Number of nodes	*Number of cores*	*Execution time in minutes*	*Parallel efficiency*
16	448	5567.2	1.0000
32	896	2900.7	0.9596
48	1344	2044.3	0.9078
64	1792	1590.0	0.8753

Execution time obtained at 16 nodes (448 cores) is used as the base.

NCBI, National Center for Biotechnology Information; PCJ-BLAST, Parallel Computing in Java-Basic Local Alignment Search Tool.

compared with no HT execution. The performance data presented in Figure 9 show good scaling up to the 128 nodes (6144 cores). The performance results at Cray XC40 have been obtained, and for the first 12,288 sequences out of 1 million read long input.

The parallelization efficiency as presented in Table 2 is over 90% up to 32 nodes. For 64 nodes is over 83% and then starts to decrease much faster. The parallel postprocessing of the XML results made by PCJ-BLAST is very efficient. The limit of I/O capabilities caused the lower performance of the XML processing on the 128 nodes.

We have compared performance with the HPC implementation of NCBI-BLAST based on splitting the query string and database into parts and running multiple jobs managed by the queuing systems as a job array (Mikailov et al., 2017). We have divided the database into 32 roughly equal blocks. The query string was divided into equal parts to the number of threads divided by the number of database parts. The experiments were run using the same input and the same BLAST parameters. The results presented in Table 2 show good scalability, but execution time is more than two times higher than for PCJ-BLAST. Such large difference comes mainly from the fact that the PCJ-BLAST implements dynamic load balancing mechanisms, which compensate for the different time required to process the query parts.

5. CONCLUSION

In this article, we have presented massively parallel execution of the sequence search, which is a key element of the NGS results processing. Usage of the nonmodified NCBI-BLAST is an additional advantage.

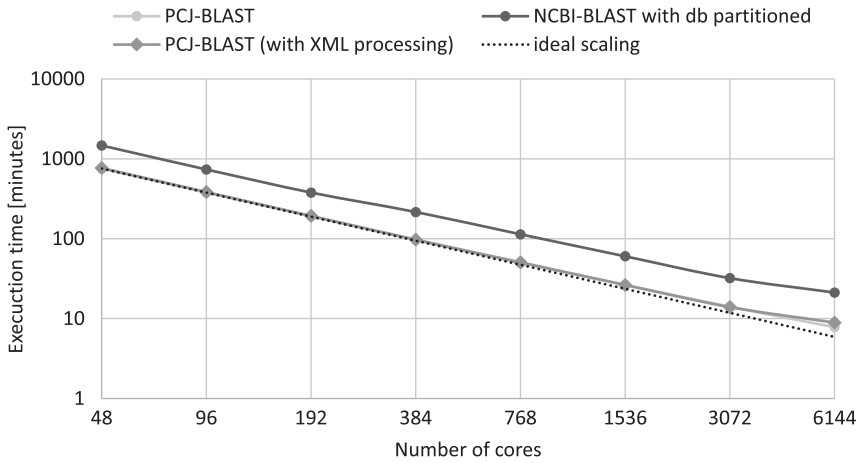

FIG. 9. Performance of PCJ-BLAST on the Cray XC40. The ideal scaling is plotted as the dotted line for reference. The scaling is based on the single node running 4 NCBI-BLAST instances executed using 12 threads each. The results are presented for PCJ-BLAST with and without XML output processing and for the NCBI-BLAST with query and database partitioning.

Table 2. Parallel Efficiency of PCJ-BLAST Running on Cray XC40

		NCBI-BLAST	*PCJ-BLAST (this work)*			
		db splitted to four blocks	*Four NCBI-BLAST per node*		*Four NCBI-BLAST per node with XML processing*	
Number of nodes	*Number of cores*	*Execution time in minutes*	*Execution time in minutes*	*Parallel efficiency*	*Execution time in minutes*	*Parallel efficiency*
1	48	1470.5	756.0	1.000	768.6	1.000
2	96	736.4	379.0	0.997	384.5	0.999
4	192	378.8	190.8	0.990	193.2	0.994
8	384	216.1	96.3	0.981	97.5	0.985
16	768	113.7	49.5	0.955	50.7	0.947
32	1536	60.3	26.4	0.893	26.4	0.909
64	3072	32.2	14.1	0.837	13.9	0.861
128	6144	21.2	7.8	0.755	8.9	0.672

Execution time obtained at single node (24 cores with hyperthreading) is used as the base. The execution time with the database and query partitioned (Mikailov et al., 2017) are presented for comparison.

The parallelization was done by work distribution based on the partitioning of the input sequence with monitoring of the execution of BLAST instances to load balance the work. The parallelization schema has been implemented in Java using the PCJ library and successfully executed on the HPC cluster and a supercomputer. Arisen I/O bottlenecks while reading reference sequence database have also been eliminated.

We confirmed that these design principles allow the application to scale almost linearly for HPC cluster, that is, more than 87% efficiency for up to 64 nodes (1792 cores) in comparison to 16 nodes (448 cores). The performance results for Cray XC40 are similar and present at least 90% parallel efficiency for 32 nodes (1536 cores) with output XML processing, and 75% parallel efficiency at 128 nodes (6144 cores) without XML processing.

The design and implementation of the application using Java with the PCJ library were fast and efficient and resulted in the preparation of the scalable application in a short time, meaning reduced development costs. The dynamic load balancing allowed for equal node utilization resulting in twofold speedup compared with other solutions based on static work distribution. The obtained results also confirm the ability of the PCJ library to parallelize large-scale applications.

As the last word, we would like to focus on future work. We have gained preliminary results, which showed that it is also possible to run PCJ-BLAST application not only on HPC clusters or supercomputers, but also on Apache Spark (https://spark.apache.org; accessed March 1, 2018) cluster. However, this is still an ongoing work and subject to another publication.

ACKNOWLEDGMENTS

The authors would like to thank CHIST-ERA consortium for financial support under HPDCJ project (Polish part funded by NCN grant 2014/14/Z/ST6/00007) and NordForsk for the support within NIASC consortium. The performance tests have been performed using ICM University of Warsaw computational facilities. This research was carried out with the support of the Interdisciplinary Center for Mathematical and Computational Modeling (ICM), University of Warsaw under grants no GB65-15 and GA69-19.

AUTHOR DISCLOSURE STATEMENT

The authors declare that no competing financial interests exist.

REFERENCES

Altschul, S.F., Gish, W., Miller, W., et al. 1990. Basic local alignment search tool. *J. Mol. Biol.* 215, 403–410.

Altschul, S.F., Madden, T.L., Schäffer, A.A., et al. 1997. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Res*. 25, 3389–3402.

Bjornson, R.D., Sherman, A., Weston, S.B., et al. 2002. TurboBLAST(r): A parallel implementation of BLAST built on the TurboHub, 0183. *In International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. Ft. Lauderdale, FL.

Braun, R., Pedretti, K.T., Casavant, T.L., et al. 2001. Parallelization of local BLAST service on workstation clusters. *Future Gener. Comput. Syst.* 17, 745–754.

Carlson, W.W., Draper, J.M., Culler, D.E., et al. 1999. *Introduction to UPC and Language Specification* (Technical Report No. CCS-TR-99-157). IDA Center for Computing Sciences. Bowie, Maryland.

Celesti, A., Celesti, F., Fazio, M., et al. 2017. Are next-generation sequencing tools ready for the cloud? *Trends Biotechnol.* 35, 486–489.

Chi, E.H.-H., Shoop, E., Carlis, J., et al. 1997. *Efficiency of Shared-Memory Multiprocessors for a Genetic Sequence Similarity Search Algorithm* (Technical Report No. TR97-05). University of Minnesota, CS Department. Minneapolis, MN.

Cofer, H. 2012. *SGI® High Throughput Computing (HTC) Wrapper Program for Bioinformatics on SGI ICE™ and SGI UV™ Systems* (Technical Report). Silicon Graphics International. Fremont, CA.

Darling, A.E., Carey, L., and Feng, W.C. 2003. *The Design, Implementation, and Evaluation of mpiBLAST* (Technical Report). Los Alamos National Laboratory. Los Alamos, NM

Hilfinger, P.N., Bonachea, D.O., Datta, K., et al. 2005. *Titanium Language Reference Manual, Version 2.19* (Technical Report No. UCB/EECS-2005-15). EECS Department, University of California, Berkeley.

Lin, H., Ma, X., Chandramohan, P., et al. 2005. Efficient data access for parallel BLAST, 72b. *In 19th IEEE International Parallel and Distributed Processing Symposium*. IEEE. Denver, CO.

Mathog, D.R. 2003. Parallel BLAST on split databases. *Bioinformatics* 19, 1865–1866.

Mikailov, M., Luo, F.-J., Barkley, S., et al. 2017. Scaling bioinformatics applications on HPC. *BMC Bioinformatics* 18, 501.

Nowicki, M., Bzhalava, D., and Bała, P. 2017. Massively parallel sequence alignment with BLAST through work distribution implemented using PCJ library, 503–512. *In International Conference on Algorithms and Architectures for Parallel Processing*. *ICA3PP 2017*. Springer International Publishing. Helsinki, Finland.

Nowicki, M., Górski, Ł., Grabarczyk, P., and Bała, P. 2014. PCJ—Java library for high performance computing in PGAS model, 202–209. *In 2014 International Conference on High Performance Computing Simulation (HPCS)*. IEEE. Bologna, Italy.

Nowicki, M., Ryczkowska, M., Górski, Ł., et al. 2016. PCJ—A Java library for heterogenous parallel computing, 66–72. *In* Zhuang, X., ed. *Recent Advances in Information Science (Recent Advances in Computer Engineering Series vol. 36)*. WSEAS Press. Barcelona, Spain.

Numrich, R.W., and Reid, J. 1998. Co-array Fortran for Parallel Programming. *SIGPLAN Fortran Forum* 17, 1–31.

Seetharam, A., Gomez, A., Purcell, C.M., et al. 2015. NCBI-BLAST programs optimization on XSEDE resources for sustainable aquaculture, 4. *In Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*. ACM. St. Louis, MO.

Sugiyama, S., and Wallace, D. 2008. Cray DVS: Data virtualization service. *In Cray User Group Annual Technical Conference*. Cray User Group, Inc., Helsinki, Finland.

Address correspondence to:
Dr. Marek Nowicki
Faculty of Mathematics and Computer Science
Nicolaus Copernicus University
Chopina 12/18
87-100 Toruń
Poland

E-mail: faramir@mat.umk.pl