CMST 20(1) 9-20 (2014) DOI:10.12921/cmst.2014.20.01.9-20

Preprocessing and Storing High-Throughput Sequencing Data

Aleksandra Świercz[1,2,*], Bartosz Bosak[3], Marek Chłopkowski[1], Arkadiusz Hoffa[1], Marta Kasprzak[1,2], Krzysztof Kurowski[3], Tomasz Piontek[3], Jacek Błażewicz[1,2]

[1] *Institute of Computing Science*
Poznan University of Technology, ul. Piotrowo 2, Poznań, Poland

[2] *Institute of Bioorganic Chemistry*
Polish Academy of Sciences, ul. Noskowskiego 12/14, Poznań, Poland

[3] *Poznań Supercomputing and Networking Center*
ul. Noskowskiego 10, Poznań, Poland

[*] *E-mail: aswiercz@cs.put.poznan.pl*

Received: 27 November 2013; revised: 9 February 2014; accepted: 10 February 2014; published online: 18 March 2014

Abstract: DNA sequencing is a process of recognizing DNA sequences of genomes. The process consists in reading short sequences, that are subsequences of a genome, and merging them into longer sequences, preferably the whole genome. In the first phase even billions of short sequences are read at once. To simplify and speed up the second phase, we develop a pipeline of preprocessing the initial set of short sequences that is removing low quality reads and duplicated reads. We also propose a method for preliminary joining overlapping sequences, which resulted in decreasing the cardinality of initial sets to 13.9% and 27.8%. We also examine possible ways to store the huge amount of experimental data. We compare different compression methods, from which the best appeared to be DSRC, developed for special type of text files containing short sequences and their quality. We test the parameters for TCP data transferring to find the best transfer rate.

Key words: DNA sequencing, compression, preprocessing, transfer optimization

I. INTRODUCTION

Recognizing chains of nucleic acids: deoxyribonucleic acid (DNA) and ribonucleic acid (RNA) consists in reading the sequence of their molecules called nucleotides. The nucleotides are symbolically labelled with letters A, C, G and T (or U) and their sequences represent genetic information of an organism. The process of reading nucleotides, called *sequencing*, is performed with biochemical instruments (*sequencers*). With technological progress the next-generation sequencers have appeared, which are able to read in parallel simultaneously billions of short reads (DNA/RNA fragments) in one run. In order to recognize the whole genome, one has to merge these short reads into longer sequences (*contigs*) covering the genome. The reasons for obtaining several contigs instead of one continuous sequence are low genome coverage in some regions, sequencing errors, and characteristics of assembly algorithms. In case the genome has not been known yet, the process of merging the reads is called *de novo* assembling, can be considered as a more complex variant of the shortest common superstring problem [1] and is computationally hard. Computation time of the assembling problem can be significantly reduced if the preprocessing stage is applied, where a number of reads of low quality are rejected.

There are a series of approaches for assembling *de novo*: string-based algorithms which are mainly used for smaller genomes [2-4], and graph-based ones that are designed for complex genomes [5-9]. In case of the latter we may distinguish methods based on so-called *de Bruijn* graphs and methods realizing the overlap-layout-consensus strategy. In the *de Bruijn* graphs nodes represent k-mers (words of k nucleotides, k much shorter than the length of reads) [6, 8]. Reads in such a graph are represented as paths through

the graph. This approach saves memory due to a constant graph size; however, we lose the information about the reads (i.e. about the original order of *k*-mers) and thus the possibility of misassembly increases. On the other hand, the overlap-layout-consensus approaches [9] keep the information about reads which are represented as nodes in the *overlap graph* (the graph representing the relation of overlapping between reads), but as a result memory usage increases. Construction of a graph then requires calculation of alignments for all pairs of nodes (reads) from the instance. Due to a huge number of reads in an instance it is impossible to determine that in an exact manner. Thus, a fast heuristic is needed, which selects *promising pairs* of reads overlapping with high probability. For the selected pairs alignments are calculated and arcs are added to the graph where longest paths represent most valuable contigs.

A large amount of experimental data coming from the next generation sequencer, Illumina Genome Analyzer GAIIx, imposes strong necessity of efficient processing, transferring and archiving the data. Such sequencer is owned by the European Center of Bioinformatics and Genomics (ECBiG), a consortium of Poznan University of Technology and Institute of Bioorganic Chemistry, Polish Academy of Sciences. Usually the output of one experiment is of several hundreds GB. It consists of text files which contain reads together with their quality expressed numerically, and configuration files which are necessary to reproduce the whole experiment and repeat analyses. In the first part of the article we describe a possible pipeline for data preprocessing, which results in deleting low-quality reads, compressing reads, and preliminary merging of overlapping reads. Our preprocessing procedure allows for more efficient computing at the assembling stage.

Another problem is archiving the experimental data for further processing, especially if one plans to start again an analysis from raw data. The problem can be divided into steps: data compression/decompression, transfer and storage. Data can be compressed in two ways: text data only or the full set of files from the experiment. In the second part of the paper we present results of our work with several compressing algorithms, both specialized and of general purpose. Effective compression is necessary for efficient transfer, thus the quality and time of the compression is crucial. We have also tested a number of protocols for transferring the data to a storage server. For the storage we have chosen the Service Platform for E-Science PLATON-U4, where Polish scientists and Polish public institutions can safely store important data.

II. PREPROCESSING OF DATA FOR *DE NOVO* ASSEMBLY

Preprocessing, as a step of the DNA assembly process, can significantly shorten overall computation time of the latter. Performed cautiously, it may result in considerable reduction of a volume of sequencing data, but without visible loss of quality of final contigs. This step consists mainly in purifying raw sequencing data by correcting, truncating or removing reads of low quality. The quality of reads can be given as a numerical data file provided by a sequencer, but it can also be estimated on the basis of nucleotide composition of reads, especially when N's are present. The preprocessing step, in addition to the data purification, may be supplemented with other procedures useful for further sequence assembly. In our approach such a procedure, leading to collecting so-called promising pairs of reads, has been applied. Our procedure is somewhat in the opposite to the well-known approach often applied at this step, which involves the dynamic programming algorithm of Smith and Waterman [10].

The preprocessing algorithm that we propose takes at the input a FASTA/FASTQ file [11-12] which is next processed according to the following sequence of procedures.

1) Rejection of reads with unspecified nucleotides (N's)
2) Addition of reverse complements of remaining reads
3) Data compression
4) Lexicographical sorting
5) Removal of duplicates
6) Δ-reads generating and sorting
7) Analysis of overlaps
8) Search for promising pairs

In detail, these operations proceed as follows. Input reads are removed from further processing if they contain unspecified nucleotides, but one should be aware that it is not appropriate for any possible data source, where N's can appear in most of reads. The addition of reverse complementary counterparts of reads is a usual step which allows for representing data possibly coming from the other DNA strand. As a result, we obtain a doubled set of contigs at the end. Due to data compression, where every base is represented with the use of two bits, the volume of data is reduced fourfold. Note that we can apply such encoding model only in the case when reads are composed of specified nucleotides solely (A, C, G, T), which we have guaranteed after the second step. Further data reduction is obtained by leaving only one representative of multiplicated reads, which happens after sorting the data lexicographically.

```
CCATGTCAACCTC        (offset = 0)
CATGTCAACCTC
ATGTCAACCTC
TGTCAACCTC
GTCAACCTC
TCAACCTC             (offset = 5)
```

Fig. 1. The set of 6 Δ-reads generated for the read CCATGTCAACCTC and the maximal offset $\Delta = 5$

Δ-reads are variants of reads with several offsets assumed. Δ is a parameter meaning the maximal offset.

For one read, Δ+1 its Δ-reads are generated, and they are suffixes of the read starting at its i^{th} nucleotide, $i \in \langle 1; \Delta + 1 \rangle$, with the offset equal to i-1 (Fig. 1).

After the generation of Δ-reads for all reads present at this step, Δ-reads are sorted lexicographically, and that gives us information which ones are prefixes of the others. A pair of Δ-reads, where one element is a prefix of the other having offset = 0, corresponds to a pair of reads overlapping without mismatches, with a shift between 1 and Δ. Such technical realization of the procedure of searching for ideal overlaps better fits to the architecture of GPU. It allows us to use a simple matching algorithm and compare only the reads that can overlap.

As one can easily imagine, coupling all matching reads on the way to selecting optimal junctions cannot end successfully. Even after some efficient data filtering, the remaining reads are too numerous and only selected pairs can be considered for their overlap. The answer which pairs (promising pairs) should be considered can be found in several ways, but we propose the following procedure. From possible covers of Δ-reads found in the previous step we select only the best ones, i.e. such pairs where one Δ-read is a longest prefix (from among the present ones) of the other Δ-read of zero offset. If there are two or more longest prefixes for a read, all of them are kept in memory. All the promising pairs of reads are the output of the preprocessing algorithm and next handled by the assembly algorithm.

Although all the procedures contribute to the reduction of majority of initial data, the algorithm would be not so useful when implemented entirely for CPU. GPGPU computing significantly shortens program execution time, but not every procedure may be implemented for that purpose. We encoded for GPU procedures realizing steps 3-7. We had to divide huge input data sets into chunks to make them possible to be processed in GPU memory. The chunk size depends on GPU memory limit set as the program parameter. For the limit equal to 1024 MB, the chunk size is about 450 MB. The remaining part of memory is used for storing sorting data (keys and indexes) and results. For data sorting we used the SRTS algorithm described in [13]. We performed $m/8$ key-value MSB radix sort steps starting from the end of read (by assigning 4 bytes to *key* field of sorted record, *value* field holds read index). After the sorting step we performed a parallel comparison between every two neighboring records, one index per a GPU thread. Results are stored in a file which is then used for pair searching (step 8).

Pre-assembly

Although very helpful in further computations when an assembly algorithm is able to take advantage of information of such kind, the set of promising pairs at the output of our preprocessing sometimes cannot be used. Autonomous assemblers usually accept at the input only a set of nucleotide sequences, possibly of different lengths. Thus we provided the option of building preliminary contigs by aggregating promising pairs into bigger structures. The optional step of the preprocessing algorithm is as follows:

9) Building preliminary contigs

The step consists in augmenting unambiguous paths, starting from pairs of exactly overlapping reads, in two directions (left and right) as long as there are single neighbors to attach. The building of a contig is stopped at a first branching point (a contig is reported at the output even if the branching takes place after one read). Once used, reads are not further taken into account, except the following case. In order to make the final assembly easier, the branching points are also encoded as preliminary contigs: junction contigs.

The preprocessing algorithm can be run in two modes: SR, in which the output contains single reads and preliminary contigs, and NSR, where only preliminary contigs are provided.

Experimental results

In order to verify efficiency of the preprocessing algorithm, we performed tests on both artificial and real data sets. The artificial instance was generated on the basis of a real DNA sequence by cutting off overlapping reads of lengths 50, with shifts between neighboring reads from 1 to 10 nucleotides. No errors were introduced into the file. The instances coming from real experiments with an Illumina sequencer contained natural errors and represented genomes of *Escherichia coli* and *Caenorhabditis elegans*. The experimental data were obtained from the Short Read Archive [14]. The data for *E. coli* (ERR022075_1) contained 22720100 reads of length 100 and, to verify results, the reference genome (NC_000913.2 from NCBI database [15]) of length 4639675 nucleotides. The average depth of coverage (defined as the total nucleotide count in the input file divided by the genome size) was 484. The *C. elegans* data (SRR065390_1) contained 33808546 reads of the same length and the reference genome (WS230 from WormBase [16-17]) of length 100286070 nucleotides, with the average depth equal to 33.

In tests on the artificial data the value of Δ was set to 10 and the preprocessing algorithm supplemented with the optional step 9 returned one contig identical to the original sequence. For the real data the algorithm produced much more contigs, every one was mapped to the reference genome by Bowtie2-2.1.0 [18] using end-to-end alignment with thresholds of 1 error (of any kind) and 0 errors (perfect matches). After steps 1-5 the initial sets of reads were reduced to 32.84% (*E. coli*) and 86.02% (*C. elegans*) of their original cardinality (with reverse complementary reads). Results returned after all 9 steps are presented in Tables 1-4. We also compared our algorithm with other preprocessing methods, chosen for the reason of their high error correction quality (reported by their authors). They are the HiTEC preprocessor based on a suffix tree approach [19] and ECHO based

Tab. 1. Results of the HiTEC and our preprocessing algorithm (steps 1–9) for *E. coli* data

Δ/Source	Max contig length	Average contig length	Number of contigs	Number of junction contigs	Number of remaining single reads	Initial cardinality reduced to [%]	Number of reads used in contigs	Proc. time [s]
1	601	110	385777	106722	2858437	14.75	4410435	351
2	631	111	390673	115885	2757407	14.37	4502302	393
3	567	109	477971	119591	2678933	14.42	4577070	422
5	573	107	647962	126187	2473236	14.29	4776171	501
10	590	106	931304	141797	2088796	13.92	5145001	698
HiTEC	100	100	N/A	N/A	22455008	98.83	N/A	23782
SGA	100	100	N/A	N/A	4520772	19.90	N/A	16674

Tab. 2. Quality of reads generated with preprocessing algorithms (*E. coli*). SR/NSR are modes with/without single reads, E0/E1 stand for perfect match/1 error allowed and RAW means raw reads from ERR022075_1 dataset

Δ/Source	Reads mapped E0 NSR [%]	Reads mapped E0 SR [%]	Reads mapped E1 NSR [%]	Reads mapped E1 SR [%]	Coverage E0 NSR [%]	Coverage E0 SR [%]	Coverage E1 NSR [%]	Coverage E1 SR [%]
1	70.190	11.080	98.740	62.920	99.835	99.915	99.885	99.977
2	46.850	9.780	96.300	62.010	96.465	97.635	99.853	99.977
3	36.650	11.010	94.200	62.210	94.042	96.325	99.905	99.977
5	26.500	11.620	90.330	61.980	92.134	95.142	99.930	99.977
10	17.970	11.620	82.460	61.140	90.810	94.171	99.945	99.975
RAW	N/A	85.320	N/A	93.730	N/A	99.974	N/A	99.977
HiTEC	N/A	98.980	N/A	99.170	N/A	99.964	N/A	99.972
SGA	N/A	96.210	N/A	99.530	N/A	99.945	N/A	99.945

on a multiple sequence alignment (MSA) approach [20]. We have also included the String Graph Assembler [21] because its early processing stages called preprocess, index, correct and filter can be run separately (without running the assembly process).

Tests were run on a desktop computer running Windows 7 x64/Linux x64 (the latter for HiTEC and ECHO testing) with Intel Core i7-3820 @ 3.7 GHz, 16 GB of RAM @ 1866 MHz and NVIDIA Geforce GTX 670 with 2 GB of global memory.

The average contig lengths are related to regular contigs, not for the junction ones. The reduction of initial cardinality has been calculated as the sum of numbers of contigs, junction contigs and remaining single reads divided by the number of input reads. The reduction will be even more efficient if single reads are not reported at the output; they do not fit well. Actually, we checked the impact of such further reduction and we got very high coverage of the reference genome only by regular contigs. For our algorithm, for $\Delta \geq 3$, we obtained over 99.9% of the genome coverage without using single reads, and the reduction of initial cardinality below 5% (for all cases). The program consumed less than 1 GB of RAM during all stages except building contigs, where it used 2.9 GB (mostly for caching reads in memory for faster access, for *C. elegans* consumed memory was less than 4 GB). Detailed results of alignment of reads and contigs are presented in Tab. 2. The HiTEC preprocessor was occupying 10 GB of RAM for over 6 hours, reduced cardinality only to 98.83% (although it is not intended to reduce dataset size) and achieved comparable quality. ECHO consumed up to 25 GB of memory and after almost 9 hours of processing was killed by the system on the *NeighborJoin* phase. SGA pipeline (preprocess, index, correct and filter) was performed using below 2 GB of RAM and took almost 5 hours.

Although the reduction obtained in tests on the data of *C. elegans* is not as impressive as before, it is quite satisfying for bigger values of Δ (Tab. 3). The bigger the Δ, the higher the possibility that contigs are incorrectly assembled. However, our additional tests verifying the genome coverage by NSR-type output (contigs without single reads) have shown that in the case of Δ=10 it is equal to 98%, where the cardinality of the dataset is equal to 8.8% of its initial state. It should be mentioned that the size of a reduced dataset is crucial for assembling algorithms used after the preprocessing stage. Huge datasets make the assemblers fail due to enormous memory usage. Tab. 4 shows detailed quality information. Among other programs only SGA was able to handle provided data. It used less than 2 GB of RAM and over 7 hours of CPU time. Unfortunately, other preproces-

Tab. 3. Results of our preprocessing algorithm (steps 1–9) for *C. elegans* data

Δ	Max contig length	Average contig length	Number of contigs	Number of junction contigs	Number of remaining single reads	Initial cardinality reduced to [%]	Number of reads used in contigs	Proc. time [s]
1	288	101	3906758	30345	19347015	68.87	9369485	823
2	287	102	5086154	49852	14396441	57.77	14300553	950
3	377	104	5215910	68658	11512954	49.68	17165235	1100
5	610	108	4487041	102393	8581102	38.96	20063352	1436
10	1066	124	2812086	162436	6411174	27.76	22173237	2207
SGA	100	100	N/A	N/A	21777343	64.41	N/A	27446

Tab. 4. Quality of reads generated with preprocessing algorithms (*C. elegans*). SR/NSR are modes with/without single reads, E0/E1 stand for perfect match/1 error allowed and RAW means raw reads from SRR065390_1 dataset

Δ/Source	Reads mapped E0 NSR [%]	Reads mapped E0 SR [%]	Reads mapped E1 NSR [%]	Reads mapped E1 SR [%]	Coverage E0 NSR [%]	Coverage E0 SR [%]	Coverage E1 NSR [%]	Coverage E1 SR [%]
1	93.960	66.410	95.920	77.670	92.798	99.609	93.096	99.842
2	92.950	60.810	95.750	74.090	96.502	99.520	96.827	99.830
3	91.120	55.120	95.240	70.460	97.452	99.432	97.831	99.816
5	85.180	43.960	93.270	63.330	97.886	99.180	98.460	99.781
10	59.040	23.610	82.490	50.070	93.803	95.749	98.061	99.588
RAW	N/A	73.700	N/A	82.400	N/A	99.759	N/A	99.861
SGA	N/A	94.59	N/A	95.450	N/A	99.500	N/A	99.552

sors did not finish processing. HiTEC consumed over 65 GB of memory and we stopped it after over 1000 minutes (of CPU time) when it was still processing the first iteration (out of nine) in the first processing split (out of six). We could not get ECHO to process the whole pipeline, either.

Our experiment and results provided by authors of other preprocessors point out that the main difference between algorithm presented in this paper and current preprocessors is the output cardinality reduction. The mentioned preprocessors correct data quality (and produce more "perfect" reads) but do not reduce input size (except removing ambiguous reads). However, in contrast to specialized preprocessing programs, the SGA Assembler pipeline [21] can reduce input size significantly (close to our method run with $\Delta = 1$). While SGA produce more perfect matching reads, the overall coverage (E1) is lower than that achieved in our method. Moreover, our method performed over 12 times faster for *C. elegans* data and almost 24 times faster for *E. coli*.

III. COMPRESSION OF SEQUENCING DATA

Next-generation sequencers produce a huge amount of experimental data, thus users are obliged to regularly move the generated files to some external disk space in order to release the server for further experiments. Raw data from experiments are used mostly at the beginning for producing text files in the FASTQ format, containing reads with qualities of nucleotides. Further analysis, like for example preprocessing, resequencing, assembling *de novo*, or microRNA analysis, needs only FASTA/ FASTQ files. For most of the experiments no other attempt to re-run any analysis involving raw data is made. Thus, it seems quite reasonable to compress raw data and FASTQ files separately, because the latter are used much more often. In this section we analyze different tools available for compressing both types of data, and in the next section we describe protocols transferring data to a storage server.

We started the compression with putting the files created in the sequencing process, together with the original directory structure, into one single file with the TAR tool. Next, the compression was performed using gzip or bzip. We used it interchangeably due to the fact that the archive size was similar. The quality of compression is presented in Tab. 5.

The raw data are composed of several types of binary and text files. As we can see in Tab. 5, the efficiency of the compression is not very high. In our further comparison of compression quality of different programs we focused on FASTQ files, for which specialized algorithms exist. Below we shortly describe tested tools.

1) Gzip (3,6,9 data compression level) is an open compression standard. This method is based on the *deflate* algorithm, a combination of Lempel-Ziv [22-23] and Huffman trees. Gzip is based on the consolidated TAR archives [24-25]. The deflate algorithm and gzip file

Tab. 5. The size of archive before and after compressing

Sequencing data folder (run)	Original size (raw data) [GB]	Size after compression [GB]	Compression ratio [%]
110718_HWUSI-EAS1865_00001_FC/	271	217	80
111104_HWUSI-EAS1865_00002_FC/	382	299	78
120228_HWUSI-EAS1865_00005_FC/	412	326	79
120312_HWUSI-EAS1865_00006_FC/	340	276	81
120412_HWUSI-EAS1865_00007_FC/	247	176	71
120423_HWUSI-EAS1865_00008_FC/	408	338	83
120511_HWUSI-EAS1865_00009_FC/	142	111	78
120803_HWUSI-EAS1865_00012_FC/	778	554	71
121122_HWUSI-EAS1865_00013_FC/	409	256	63
121210_HWUSI-EAS1865_00014_FC/	591	423	72
121218_HWUSI-EAS1865_00015_FC/	95	77	81
130208_HWUSI-EAS1865_00016_FC/	293	241	82
AZ-PJ_120712_HWUSI-EAS1865_00011_FC/	428	252	59
AZ-TT-JG02-62Y44AAXX/	347	303	87

format have been standardized in several RFC documents [26-28].

2) Bzip2 (3,6,9 data compression level). Input data are divided into blocks, and each block is subjected to compression separately. The compression process consists of several parts and to each block the following algorithms are applied: the Burrows-Wheeler transformation [29], *Move to Front* and Huffman algorithm [30].
3) 7-zip [31] (standard and LZMA algorithm). The base and default algorithm is LZMA, which is an improved and optimized version of the Lempel-Ziv algorithm. This program also allows to use the standard *deflate* Lempel-Ziv method.
4) DSRC [32]. This program is a specialized compression algorithm designed to compress the files in the FASTQ format. Authors noticed that a FASTQ file can be divided into three separate parts and it is possible to create separate data streams. The streams are processed almost independently by the algorithm. The hierarchical structure of the data allows for quick and easy access to data records. Input data are divided into blocks consisting of records (default 32), and these blocks are grouped into superblocks containing a specified number of blocks (default 512). This algorithm allows for an incremental creating of an output file. Each superblock is compressed independently. The program allows one to decompress a selected superblock without decompressing the entire archive.
5) KungFQ [33]. This tool is also dedicated to FASTQ files. The key step of this method is the conversion of a FASTQ file into an optimized binary format which was proposed by authors. The binary file is next compressed by standard tools (such as gzip or LZMA). This algorithm does not use additional files (for example, the reference genome) and scans the input file only once. An important advantage of the program is the fact that it has a constant memory usage. The program also allows for lossy compression.

In Tab. 6 the programs described above are compared. For given five different datasets we present sizes of compressed FASTQ files, and compression and decompression time. It can be seen that the general-purpose programs, such as gzip, bzip, and 7-Zip, have much longer compression time and a similar or worse compression degree than the tested specialized algorithms. It is worth mentioning that the general-purpose programs allow to compress files of different formats, so they have much wider potential application. However, the nature of our work (DNA/RNA *de novo* sequencing) and the fact that the main and most commonly used file format is FASTQ direct our future work to designing a specialized compressing algorithm, with the emphasis put on maximization of the compression degree within acceptable time.

IV. ARCHIVING DATA FROM SEQUENCER

Each launch of the Illumina sequencer generates a significant amount of bioinformatic data. In a typical case, this is several hundreds of GB of information. Due to limited capacity of local drives and insufficient mechanisms protecting integrity of the data on a local computer, the resulting data cannot remain there and must be safely and effectively transferred to a dedicated data storage space guaranteeing the requested level of the Quality of Service (QoS).

Given the nature of the handled data as part of the work, we attempted to evaluate available solutions and find the optimal backup procedure taking into account three basic elements:

- data compression
- data storage
- data transfer

Compressing the data

The data obtained from the sequencer are stored in the form of a complex structure of files and directories which are difficult to manage and transfer. Additionally, as contents of most of the files consist of text strings that are relatively easy to compress, it was natural to use methods of integration and compression in order to reduce the complexity and data volume. After collecting main requirements an analysis of popular tools was performed. The final decision was to use bzip2 (based on the Burrows-Wheeler algorithm) and tar programs. By using these tools it was possible to reduce the size of data about 30%, while the complex structure of directories and files were put in an easy-to-use single tar.bz2 archive. The compressed file containing the data from a single sequencer run, and at the same time the basic unit for archiving, takes about 100-500 GB of memory.

Storage

The target space for storing generated data was defined in the early stages of cooperation between Poznań Supercomputing and Networking Center (PSNC) and the European Center of Bioinformatics and Genomics (ECBiG). We recognized as the best solution the PLATON-U4 service, in which Polish scientists and public institutions can securely store their data [34].

Under the term PLATON-U4 there are a number of advanced technologies, from which the most important are:

1) support for multiple protocols like SFTP, WebDAV, and GridFTP,
2) a logical file system,
3) automatic data replication (synchronous and asynchronous),
4) the use of advanced storage systems: disk arrays, file servers and tape libraries,
5) encrypted data transfer and access control based on X509 certificates,
6) geographical spread of access nodes and storage in 10 cities.

For the purpose of gathering information generated by ECBiG the appropriate account was created in the PLATON-U4, for which the Service Level Agreement (SLA) contract with the list of necessary parameters was established.

Tab. 6. Comparison of quality and time of compression done by different programs

Original size [kB]	Size after compression [kB]	Compression time	Decompression time	Size after compression [kB]	Compression time	Decompression time
	gzip -3			**bzip2 -3**		
6830080	2135260	6 m 10.733 s	1 m 17.363 s	1562660	17 m 3.809 s	4 m 59.891 s
6830080	2218868	6 m 10.768 s	1 m 17.855 s	1648152	17 m 29.097 s	5 m 08.744 s
570651	182184	30.269 s	6.453 s	134168	1 m 25.156 s	25.286 s
574511	183360	30.258 s	6.551 s	134680	1 m 27.113 s	25.467 s
574466	180568	30.181 s	6.532 s	132372	1 m 26.294 s	25.228 s
	gzip -6			**bzip2 -6**		
6830080	1958420	21 m 50.977 s	1 m 10.305 s	1548636	18 m 4.346 s	5 m 15.812 s
6830080	2043904	22 m 26.364 s	1 m 08.563 s	1636176	18 m 35.641 s	5 m 24.052 s
570651	167436	1 m 50.833 s	5.830 s	133096	1 m 29.932 s	28.014 s
574511	168540	1 m 48.859 s	5.839 s	133396	1 m 32.523 s	28.008 s
574466	165856	1 m 49.540 s	5.949 s	131228	1 m 31.795 s	26.809 s
	gzip –9			**bzip2 -9**		
6830080	1913808	56 m 50.164 s	1 m 10.617 s	1538260	19 m 0.168 s	6 m 55.324 s
6830080	1998384	64 m 12.428 s	1 m 10.008 s	1628048	19 m 31.541 s	6 m 58.120 s
570651	163972	4 m 29.394 s	5.935 s	132248	1 m 34.447 s	36.021 s
574511	165072	4 m 20.572 s	6.000 s	132496	1 m 37.562 s	38.130 s
574466	162264	4 m 39.746 s	6.106 s	130324	1 m 35.978 s	35.012 s
	7-zip			**7za LZMA**		
6830080	1537704	187 m 10.121 s	2 m 59.665 s	1537692	207 m10.172 s	2 m 58.914 s
6830080	1614696	157 m 21.990 s	3 m 07.710 s	1614688	226 m27.213 s	3 m 22.438 s
570651	132204	17 m 35.781 s	15.310 s	132204	17 m 35.711 s	15.306 s
574511	132864	17 m 28.025 s	15.381 s	132864	17 m 27.914 s	15.389 s
574466	132653	17 m 32.117 s	15.149 s	130320	17 m 19.232 s	15.163 s
	DSRC			**KungFQ**		
6830080	1337214	3 m 39.238 s	2 m 21.552 s	1525520	31 m 35.303 s	371 m 00.590 s
6830080	1404890	3 m 51.110 s	2 m 50.168 s	1605988	35 m 40.876 s	375 m 38.334 s
570651	115229	18.376 s	12.062 s	131204	2 m 41.126 s	30 m 10.569 s
574511	115248	18.861 s	12.311 s	131780	2 m 40.606 s	30 m 53.394 s
574466	113376	18.668 s	11.915 s	129388	2 m 39.604 s	31 m 18.229 s

Among others, the created SLA contract specifies the limit on the size of the stored data equal to 10 TB and the principles of creating replicas.

Data transfer

The large size of the generated data volumes defines essential requirements when it comes to transfer of information between a computer located in ECBiG and storage space in the PLATON-U4 services. To reduce the complexity and length of network connections and thus increase the bandwidth, the basic action was to select the PLATON-U4 access node located as close as possible to ECBiG, in the Poznań Supercomputing and Networking Center. The following is a path of such a configured connection:

traceroute to an1-plg.storage.pionier.net.pl (150.254.173.118), 30 hops max, 40 byte packets

1) 150.254.17.129 (150.254.17.129) 1.307 ms 1.293 ms 1.312 ms
2) hellfire.put.poznan.pl (150.254.6.36) 0.360 ms 0.424 ms 0.453 ms
3) 150.254.212.212 (150.254.212.212) 1.142 ms 1.091 ms 0.881 ms
4) 150.254.255.65 (150.254.255.65) 1.132 ms 1.073 ms 0.912 ms
5) pimpstar-ext-2.man.poznan.pl (150.254.162.34) 1.874 ms 1.527 ms 1.338 ms
6) an1-plg.storage.pionier.net.pl (150.254.173.118) 2.953 ms 2.052 ms 1.979 ms

When performing basic tests, the time to transfer a 1 GB file between the selected two systems using SFTP was close to 40 s, giving a bandwidth ˜ 27 MB/s. The response time of the access point (RTT) was below 2 ms.

The attempt was made to optimize the data transfer time using the GridFTP protocol [35] and its client tool named globus-url-copy. GridFTP is an extension of the standard FTP protocol, and is dedicated to efficient transfer of large amounts of data. It offers features not available in other tools, such as:

- parallel transfer using multiple TCP streams,
- the ability to transfer data in pieces simultaneously from multiple sources (so called stripped transfer),
- transferring file fragments.

The optimization procedure aimed at finding the best possible values for the parameters corresponding to the number of connections (-p) and the TCP buffer size ($-\mathrm{tcp} - \mathrm{bs}$).

Fig. 2 shows transmission time of the file of 1 GB size for the default TCP buffer and a variable number of streams. Fig. 3 shows transfer speed noted for this situation. It can be noticed that the best values were obtained for six and nine streams. The measured data transfer time in this case was near 20 s, the communication throughput was approximately 50 MB/s. In practice, using two streams allows to achieve satisfactory results, and using a larger number than 6 streams does not result in any improvement of transmission parameters. For 50 streams a significant drop in performance probably caused by memory constraints was noted.

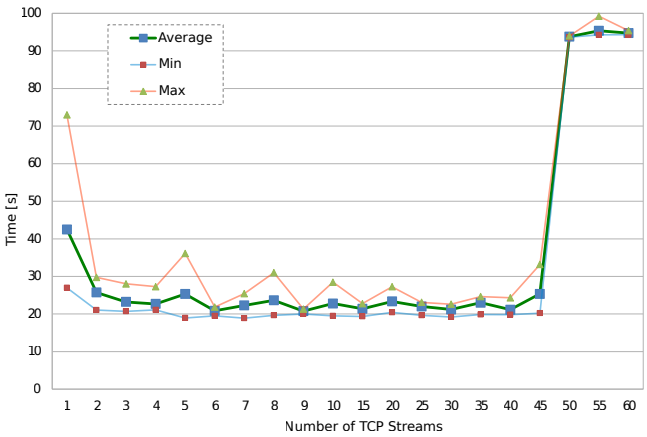


Fig. 2. Transfer time of a 1 GB file depending on the number of TCP streams

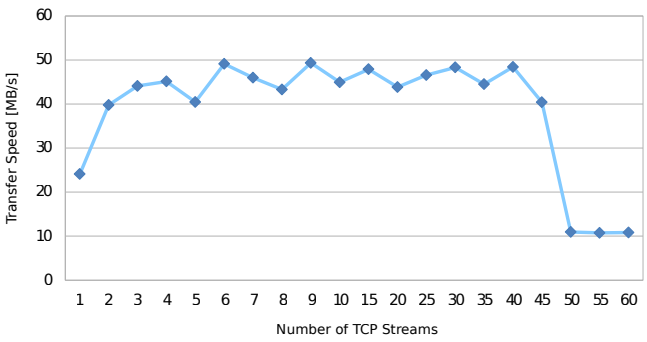


Fig. 3. Average transfer speed of a 1 GB file depending on the number of TCP streams

The next step in the optimization was to determine whether a modification to the TCP buffer size can have a positive impact on the data transfer time. In newer Linux operating systems, the size of the buffer is automatically optimized [36-37], and generally there is no need to manually adjust this parameter. However, in specific conditions the arbitrary setting of the buffer size may be advantageous.

According to the manual placed on the GridFTP website [38], the optimal size buffer can be calculated using the following formula:

$$\mathrm{tcp} - \mathrm{bs} = \text{bandwidth in Megabits per second (Mb/s)} \times \text{RTT in milliseconds (ms)} \times 1000/8.$$

As previously, the conducted RTT is ~ 2 ms. The problem is to accurately estimate the bandwidth, but it can be initially assumed that the bottleneck is the bandwidth of the network card installed on the client computer – speed of 1000 Mb/s. After substituting these values into the formula we get:

$$\mathrm{tcp} - \mathrm{bs} = 1000 \text{ Mb / s} \times 2 \text{ ms} \times 1000/8 = 250000 \text{ B}.$$

The calculated buffer size of 250 000 bytes was used as the median for the performed tests (Fig. 4).

Fig. 4. Transfer time of 1 GB file depending on the TCP buffer size

Experimental results show that manipulation of the buffer size does not reduce the transit time below the value obtained by automatic optimization done by the operating system [36]. Therefore, there is no need to manipulate with $\mathrm{tcp} - \mathrm{bs}$.

Taking into account the results of the tests presented above, we conducted 5 attempts at uploading the actual 300 GB size file using SFTP and GridFTP with 6 TCP streams. The measured transfer time via SFTP was 11783 $\pm$ 498 s, which gives an average transfer rate of 27.34 $\pm$ 1.11 MB/s. Using GridFTP the time of transfer was cut to 7851 $\pm$ 497 s and the average transfer rate was increased up to 41.03 $\pm$ 2.44 MB/s.

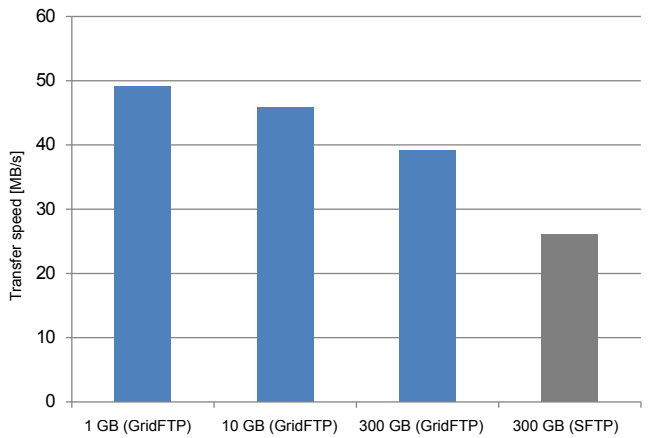


Fig. 5. Transfer speed depending on file size

On the other hand, deterioration of the transfer performance was observed for the transfer of large files. This phenomenon was confirmed by subsequent tests and is clearly visible also for 10 GB files (Fig. 5). Although the decrease of the speed is significant, GridFTP remains considerably more efficient than SFTP also for the real 300 GB files. Therefore, it is preferred for transferring such bioinformatic data.

V. CONCLUSIONS

In this article we have presented a possible pipeline for data preprocessing. This initial phase, preceding *de novo* assembly, reduces the cardinality of the input data set by removing reads with unspecified nucleotides, removing duplicated reads, compressing and sorting data, and merging reads without ambiguity in junctions into longer contigs. In case of *E. coli* dataset, the initial cardinality has been reduced to 14.8% if we allow merged reads to be shifted by one nucleotide only. By introducing Δ-reads, the method is able to find consecutive reads that are shifted by up to Δ nucleotides and merge them as well. If Δ=10, the initial cardinality has been decreased to 13.9%. Further on, if we do not take into account single reads that do not fit well, reduction of the initial set is below 5%. A slightly lower reduction rate have been obtained for *C. elegans* dataset. In case of Δ=10, the data set is equal to 27.8% of its initial state (compare Tables 1–4 for the results). The preprocessing method has been sped up with the use of GPU processors.

Another important issue presented in the article is archiving experimental data coming from the Illumina sequencer. We have shown the necessity of storing FASTA/FASTQ files separately, because they are used more often in further analysis. We have compared several compressing methods: the general usage ones, and those specialized for FASTQ files. The latter one called DSRC has appeared to have the best compression rate and the shortest compression time. Its decompression time has been comparable to other general usage compression methods. In our computational experiments we have noticed that another method specialized for FASTQ files, KungFQ, has got extremely long decompression time, far behind other methods.

We have tested the parameters corresponding to the transfer of data to the PLATON-U4 service: the number of connections and TCP buffer size. We have observed the speedup of the transfer up to 6 TCP streams. For 50 streams a significant drop in performance most likely caused by memory constraints was noted. The buffer size is automatically optimized in the Linux operating system, and manual optimization has not improved the transfer time.

Acknowledgement

The research has been supported by grants No. DEC-2011/01/B/ST6/07021 and 2012/05/B/ST6/03026 from the National Science Centre, Poland.

References

[1] T. Jiang, M. Li, *DNA sequencing and string learning*, Mathematical Systems Theory **29**, 387-405 (1996).

[2] J.C. Dohm, C. Lottaz, T. Borodina, H. Himmelbauer, *SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing*, Genome Res **17**, 1697-1706 (2007).

[3] D.W. Bryant, W.K. Wong Jr., T.C. Mockler, *QSRA: a quality-value guided de novo short read assembler*, BMC Bioinformatics **10**, 69 (2009).

[4] R.L. Warren, G.G. Sutton, S.J. Jones, R.A. Holt, *Assembling millions of short DNA sequences using SSAKE*, Bioinformatics **23**, 500-501 (2007).

[5] R. Li, W. Fan, G. Tian, H. Zhu, L. He, et al., *The sequence and de novo assembly of the giant panda genome*. Nature **463**, 311-317 (2010).

[6] J.T. Simpson, K. Wong, S.D. Jackman, J.E. Schein, S.J. Jones, et al., *ABySS: a parallel assembler for short read sequence data*, Genome Res **19**, 1117-1123 (2009).

[7] Li R, Zhu H, Ruan J, Qian W, Fang X, et al., *De novo assembly of human genomes with massively parallel short read sequencing*, Genome Res **20**, 265-272 (2010).

[8] D.R. Zerbino, E. Birney, *Velvet: algorithms for de novo short read assembly using de Bruijn graphs*, Genome Res **18**, 821-829 (2008).

[9] J. Blazewicz, M. Figlerowicz, P. Gawron, M. Kasprzak, E. Kirton, D. Platt, A. Swiercz, L. Szajkowski, *Whole genome assembly from 454 sequencing output via modified DNA graph concept*, Computational Biology and Chemistry **33**, 224-230 (2009).

[10] T.F. Smith, M.S. Waterman, *Identification of common molecular subsequences*, Journal of Molecular Biology **147**, 195-197 (1981).

[11] http://www.bioinformatics.nl/tools/crab_fasta.html.

[12] P.J.A. Cock, C.J. Fields, N. Goto, M.L. Heuer, P.M. Rice. *The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants*, Nucleic Acids Res **38**, 1767-1771 (2010).

[13] D. Merrill, A. Grimshaw, *Revisiting sorting for GPGPU stream architectures*, Technical Report CS2010-03, University of Virginia, Department of Computer Science, Charlottesville, VA, USA, (2010).

[14] Sequence Read Archive (http://www.ncbi.nlm.nih.gov/sra).

[15] Escherichia coli str. K-12 substr. MG1655, complete genome (http://www.ncbi.nlm.nih.gov/nuccore/NC_000913.2).

[16] WormBase, ftp://ftp.wormbase.org/pub/wormbase/releases/WS230/species/c_elegans/ c_elegans.WS230.genomic.fa.gz.

[17] K. Yook, T.W. Harris, T. Bieri, et al., *WormBase 2012: More genomes, more data, new website*, Nucleic Acids Research **40**, D735-D741 (2012).

[18] B. Langmead, S. Salzberg, *Fast gapped-read alignment with Bowtie 2*, Nature Methods **9**, 357-359 (2012).

[19] L. Ilie, F. Fazayeli, S. Ilie, *HiTEC: accurate error correction in high-throughput sequencing data*, Bioinformatics **27**, 295-302 (2011).

[20] W.C. Kao, A.H. Chan, Y.S. Song, *ECHO: a reference-free short-read error correction algorithm*, Genome Res **21**, 1181-1192 (2011).

[21] J.T. Simpson, R. Durbin, *Efficient de novo assembly of large genomes using compressed data structures*, Genome Res **22**, 549-556 (2012).

[22] J. Ziv, A. Lempel, *A Universal Algorithm for Sequential Data Compression*, IEEE Transactions on Information Theory **23**, 337-343 (1977).

[23] J. Ziv, A. Lempel, *Compression of individual sequences via variable rate coding*, IEEE Transactions on Information Theory **24**, 530-535 (1978).

[24] Deflate and inflate algorithms (http://www.gzip.org/algorithm.txt).

[25] J-L Gailly, GNU gzip, 2013 (http://www.gnu.org/software/gzip/manual/gzip.pdf).

[26] P. Deutsch, J-L. Gailly, ZLIB Compressed Data Format Specification version 3.3, 1996 (http://tools.ietf.org/pdf/rfc1950.pdf).

[27] P. Deutsch, DEFLATE Compressed Data Format Specification version 1.3, 1996 (http://tools.ietf.org/pdf/rfc1951.pdf).

[28] P. Deutsch, GZIP file format specification version 4.3, 1996 (http://tools.ietf.org/pdf/rfc1952.pdf).

[29] M. Burrows, D.J. Wheeler, *A block sorting lossless data compression algorithm*, Technical Report 124, Digital Equipment Corporation. (1994).

[30] D.A. Huffman, *A Method for the Construction of Minimum-Redundancy Codes*, Proceedings of the I.R.E., 1098-1102 (1952).

[31] 7-zip file archiver (http://www.7-zip.org/).

[32] S. Deorowicz, S. Grabowski, *Compression of DNA sequence reads in FASTQ format*, Bioinformatics **27**, 860-862 (2011).

[33] E. Grassi, F.D. Gregorio, I Molineris. KungFQ: *A Simple and Powerful Approach to Compress fastq Files*, IEEE/ACM Transactions on Computational Biology and Bioinformatics 9, pp. 1837-1842 (2012).

[34] M. Brzezniak, N. Meyer, R. Mikołajczak, G. Jankowski, M. Jankowski. *Popular Backup/Archival Service and its Application for the Archival of the Network Traffic in the PIONIER Academic Network*, CMST Special Issue, 109-118 (2010).

[35] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, *The Globus Striped GridFTP Framework and Server*, SC'05, ACM Press, (2005).

[36] J. Semke, M. Jamshid, M. Matthew, *Automatic TCP buffer tuning*. ACM SIGCOMM Computer Communication Review **28.4**, 315-323 (1998).

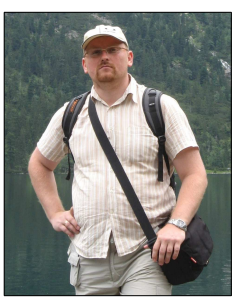
[37] Enabling High Performance Data Transfers, at Pittsburg Supercomputing Center, (http://www.psc.edu/index.php/networking/641-tcp-tune#Linux).

[38] GT 5.0.2 GridFTP (http://www.globus.org/toolkit/docs/5.0/5.0.2/data/gridftp/).

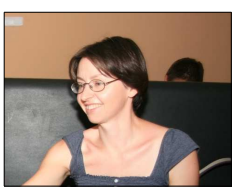
Aleksandra Świercz received her MSc degree in computer science at the Poznan Univeristy of Technology, Poland, in 2002, and defend her PhD in 2007 at the same University in the bioinformatics field. Since 2000 she has been employed at the Institute of Computing Science, Poznan University of Technology and at the Institute of Bioorganic Chemistry, Polish Academy of Sciences. She worked as a research assistant at the Hong Kong Polytechnic University. Her scientific research interests are focused on computational analysis of genomic sequences: DNA/RNA sequencing, assembling, analysis of microarray data, graph theory.

Bartosz Bosak received his MSc degree in computer science at Poznan University of Technology in Poland (Laboaratory of IT Systems in Management). Since 2007 he is working at the Application Department of Poznań Supercomputing and Networking Center as a system analyst and developer. His research interests concern Grids, HPC, communication in distributed environments and service integration in SOA. He has been involved in diverse European and national projects, including BREIN (FP6), MAPPER (FP7) and PL-Grid.

Marek Chłopkowski is a PhD student at Poznan University of Technology. In 2010 he recieved MSc in Computer Science. He is mainly interested in GPGPU applications in fields of data compression and computational biology including redesign and adaptation of serial algorithms for parallel processing. Recreations include racing, snowboarding and running.



Arkadiusz Hoffa received MSc degree in computer science at the Poznan University of Technology, Poland in 2008. He started there PhD studies at where he was an Assistant Professor in period 2008 and 2011. Arkadiusz is working as java programer in private sector and is continuing his research independently. His research interest are biological data compression and alternative splicing types identifying.



Marta Kasprzak received the MSc degree in computer science at the Poznan University of Technology, Poland, in 1995. In 2004, she defended her habilitation thesis at the same university. She has been continuously employed at the Institute of Computing Science, Poznan University of Technology since 1994. She focuses her scientific research on bioinformatics/computational biology, mainly on DNA sequencing, assembly and mapping, with the stress put on theoretical analysis of these problems: computational complexity and modeling with the use of graph theory.

Krzysztof Kurowski, holds a PhD degree in Computer Science and he is leading the Applications Department at Poznań Supercomputing and Networking Center, Poland. He was actively involved in many EU-funded R&D projects in the areas of Information Technology and High Performance Computing over the last decade, including GridLab, inteliGrid, HPC-Europa, QosCosGrid or MAPPER. He was a research visitor at University of Queensland, Argonne National Lab, and CCT Louisiana States University. His research activities are focused on the modeling of advanced applications, scheduling and resource management in heterogeneous and hierarchical computing systems, including GPU-based systems. Results of his research efforts have been successfully presented at many international conferences and workshops. He has also successfully published several journal papers and co-edited the scientific book in the area of large-scale computing.

Tomasz Piontek received his MSc in computer science in 1998 at Poznan University of Technology (Parallel and Distributed Computing). After his MSc, he joined the programmers group at PUT and worked on mobile network protocol analyzers software for Siemens A.G. and Tektronix, Inc. Since 2002 he has been working in Poznań Supercomputing And Networking Center in Application Department and was involved in many EU-funded and national R&D projects in the area of Grids and HPC Computing: GridLab, ACGT, QosCosGrid, PL-Grid, MAPPER. His research interests include distributed computing, large-scale simulations and resource management. He is an author or co-author of several papers in professional journals and conference proceedings. In PSNC he leads a team developing QosCosGrid middleware services and tools and is responsible for collaboration with domain research groups in the PLGrid PLUS project.

Jacek Błażewicz Professor and a vice-director (since 1987) of the Institute of Computing Science of Poznan University of Technology. He is also a professor at the Institute of Bioorganic Chemistry of PAS. Member of the Polish Academy of Sciences (since 2002). Member of the Polish Informatics Society and Founding Member of the Polish Bioinformatics Society (president of the Revisory Committee – since 2008); Member of the Scientific Councils of IPI PAS and IChB PAS. He is also a vice-president of the Combinatorial Optimization Working Group of EURO (European Association of Operational Research Societies) – since 1995; a Honorary Coordinator of the Computational Biology, Bioinformatics and Medicine WG of EURO - since 2007 and a Member of the EPSRC Peer Review College. Since 1997 he is an editor of the series International Handbooks on Information Systems at Springer and a member of editorial boards of 10 international journals (Parallel Computing, Journal of Heuristics, Journal of Scheduling among others). Co-chairman of 35 international conferences (among others: Dagstuhl Symposium on Scheduling – 1995, 1997, 1999, 2002, 2004; Symposium on Parallel & Distributed Processing – Aussois – 1998, Marseille – 2001, 2008; Hangzhou – 2011; EURO Conference on Computational Biology, Bioinformatics and Medicine – Rome – 2008; Nottingham – 2012). In the area of computer science and bioinformatics he published over 350 papers and books. His total citation number is over 3400 and h_index= 27 (according to ISI Citation Search). He was awarded EURO Gold Medal (1991) and the Doctorate Honoris Causa of the University of Siegen (2006). He is also a recipient of the State Award, 8 awards of the Ministry of Science and Higher Education as well as the award of Division IV of PAS. In 2012 he was elevated to the position of IEEE Fellow. In 2012 a Copernicus Prize was bestowed upon him and in 2013 he was elected for a position of Vice-President of IFORS.