K-mer Counting for Genomic Big Data

Jianqiu Ge[1,2], Ning Guo[1], Jintao Meng[1], Bingqiang Wang[1], Pavan Balaji[3], Shengzhong Feng[1], Jiaxiu Zhou[4], and Yanjie Wei[1(✉)]

[1] Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China
yj.wei@siat.ac.cn
[2] University of Science and Technology of China, Hefei 230041, China
[3] Argonne National Laboratory, Lemont, IL 60439, USA
[4] Shenzhen Children's Hospital, Shenzhen 518038, China
shirleyzjx@163.com

Abstract. Counting the abundance of all the k-mers (substrings of length k) in sequencing reads is an important step of many bioinformatics applications, including de novo assembly, error correction and multiple sequence alignment. However, processing large amount of genomic dataset (TB range) has become a bottle neck in these bioinformatics pipelines. At present, most of the k-mer counting tools are based on single node, and cannot handle the data at TB level efficiently. In this paper, we propose a new distributed method for k-mer counting with high scalability. We test our k-mer counting tool on Mira supercomputer at Argonne National Lab, the experimental results show that it can scale to 8192 cores with an efficiency of 43% when processing 2 TB simulated genome dataset with 200 billion distinct k-mers (graph size), and only 578 s is used for the whole genome statistical analysis.

Keywords: K-mer counting · Genome sequence analysis
Performance and scalability

1 Introduction

With the rapid development of Next-Generation Sequencing (NGS) technology, the growth rate of genomic data is even faster than Moore's law. Pre-analyzing the genomic data has become an essential component in many bioinformatics applications and it poses a great challenge, especially in genome assembly [1, 2]. Most of the popular parallel genome assemblers are based on de Bruijn Graph strategy, a throughout overview of the number and the distribution of all distinct k-mers provides detailed information about the size of de Bruijn graph. For this reason, it becomes increasingly important to build the histogram of frequency of each distinct k-mer so as to meet the demand of these critical bioinformatics pipelines, including genome assembly, error correction [3], multiple sequence alignment [4], metagenomic data classification and clustering.

State-of-art k-mer counting tools can be classified into two categories: one is based on hard disk and shared memory environment, the other is based on distributed memory environment.

Shared Memory Tools. Tools developed in early days are mainly classified into this category. These k-mer counting tools rely on different techniques. Firstly, the most well-known k-mer counting tool is Jellyfish [5], which uses a lock-free hash table and lock-free queues for communication so that several threads can update the hash table at the same time. KMC2 [6], MSPKmerCounter [7] and DSK [8] use disk-based partitioning technique to enable a low memory footprint with huge data processing. Methods such as BFCounter [9], Turtle [10] and KHMer [11] filter low coverage k-mers with Bloom filter to save memory consumption.

Distributed Memory Tools. As the size of genomic data increases dramatically in recent years, shared memory tools have failed to handle these data. Using high performance cluster to accelerate this procedure and breaking the limit of memory usage is a tendency. Representative tools include Kmerind [12] and Bloomfish [13]. Kmerind is the first k-mer counting and indexing library for distributed memory environment, it is implemented over MPI and contains many optimizations on efficient SIMD implementation and data structures. Bloomfish integrated Jellyfish into a MapReduce framework called Mimir [14], and 24 TB data were processed in 1 h.

In this paper, we propose a distributed k-mer counter with a higher scalability than distributed counting tools such as Bloomfish and Kmerind. The experimental results on Mira supercomputer show that, with 8192 cores, it can process 2 TB simulated genomic data with 200 billion distinct k-mers (graph size) in approximately 578 s.

2 System Design and Implementation

Our proposed k-mer counter contains three components, parallel I/O, k-mer extraction and distribution, counting. These components are illustrated in Fig. 1 and will be introduced in the following. For speeding up the pipeline during implementation, these three phases are further overlapped at a high degree with data streaming technology.

Parallel I/O. Improving the loading speed for TB or even PB level of sequencing data into memory from hard drive with multiple processes faces great challenges. Many traditional genome assembler and k-mer counting tools use one thread to load data, which usually takes several hours as the data size scales to TB level.

In our work, we adopt a similar approach as SWAP2 [15] and HipMer [16], parallel I/O module is used to speed up the loading process. This module supports both FASTQ and FASTA format sequence data. Firstly, the data are divided into n virtual fragments, where n is the number of processes. Since splitting data may break the DNA reads in the middle, a location function is used to check the beginning and ending point, so that each process can quickly locate the position information and send it to neighboring process.

K-mer Extraction and Distribution. After reading the sequences from the data files, each process will extract k-mers from sequences and then distribute k-mers to their corresponding processes according to a given hash function. In the extraction phase, a sliding window of length k will be applied to break short reads to k-mers. In the distribution phase, the generated k-mers are collected and grouped with a given hash

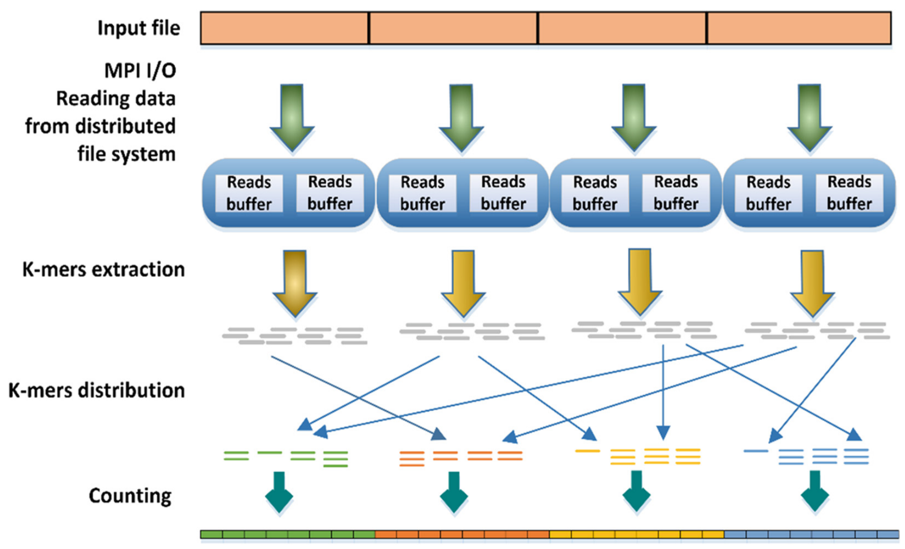

Fig. 1. Pipeline of distributed k-mer counting

function, and then each group of k-mers will be delivered to its corresponding processes with all_to_all collective communication protocols.

In this step, two other techniques are applied to further improve the computation and communication efficiency. The first one is data pre-compression and instruction level optimization on k-mer extractions. In this method, the bit operation and SSE instruction is used to further compress the computing time used in this phase. The second is to overlap the computing time in k-mer extraction phase and the communication time in k-mer distribution phases. In this method, the non-blocking all_to_all communication optimization helps to improve the efficiency.

Counting. Two types of results are counted and analyzed. One is the number of distinct k-mers, the other is frequency of each distinct k-mer. By using these basic results, almost all the other counting results can be further computed. Since the direction of DNA sequences is rarely known, genome assemblers, k-mer counting tools and many other bioinformatics applications usually treat the k-mers and their reverse complements as equivalent, we also use the same technique for our k-mer counter.

3 Experiment Setup

3.1 Dataset

To analyze the performance of k-mer counting tools, we selected two types of the datasets, real sequencing data from 1000 Genomes, and four other simulated datasets (ranging from 250G to 2 TB) generated from human reference (Downloaded from NCBI: GRCh38,

Table 1. Statistics of Experiment Datasets

	Human Genome	Dataset1	Dataset2	Dataset3	Dataset4
Size (GB)	4.5	250	500	1000	2000
Read Length (bp)	100	100	100	100	100
No. of Reads (Million)	28.8	3066.8	7133.6	14267.2	28534.4
No. of Distinct k-mers (Billion)	0.3	27.69	54.16	105.36	203.49

Data size: 3 GB). The test datasets in fasta format have both good coverage and large genome size. Detailed information for the datasets is shown in Table 1.

The Simulation Method for Generating Simulated Data. We selected human genome as the initial reference sequences in the experiment. Mutation is randomly generated in human reference sequences, simulation coverage is 50X, the length of each sequence read is 100 bp. Moreover, we also introduce 1% sequencing errors in simulated sequence reads. See Table 1 for the details of the 4 simulated datasets and human genome dataset.

3.2 Platform

In this section, we examine the performance of k-mer counting tools both on single node and high performance computing clusters. Single-node tests were conducted on a HPC server, which has Ivy Bridge-E Intel Xeon E5-2692 processor with 20 M L3 cache, 64 GB of DDR3 RAM. Distributed memory experiments were performed on IBM Blue Gene Q-Mira supercomputer at Argonne National Laboratory. Mira contains 48 cabinets, 49,152 compute nodes. Each compute node is equipped with 16 IBM PowerPC A2 processors and 16 GB RAM. All the compute nodes are connected by Infiniband, and the network of Mira follows 5D-torus architecture, the communication bandwidth is 0.9 GB/s per node. In addition, the I/O of Mira uses IBM GPFS system; it supports parallel I/O and the theoretical peak performance of its bandwidth is 32 GB per node.

4 Performance Evaluation

4.1 Single Node Test

In this section, we conducted a performance evaluation for several tools. We selected three tools, KMC2, Jellyfish and MSPKmerCounter. Both the real human sequencing data and simulated datasets are used in the experiments. During the test, we assigned one thread per core when running Jellyfish, KMC2 and MSPKmerCounter. Experimental results are shown in Tables 2 and 3.

Table 2. The time usage for human dataset using 24 cpu cores

	KMC2	Jellyfish	MSKPKmerCounter
Execution time (seconds)	27	37	62

Table 3. The time usage for simulated datasets using 24 cpu cores

	Dataset1	Dataset2	Dataset3	Dataset4
MSPKmerCounter	130 m	>6 h	out of memory	out of memory
KMC2	23 m	93 m	260 m	out of memory
Jellyfish	150 m	354 m	771 m	1495 m

As can be seen, KMC2 is the fastest k-mer counter. For GB range genomic data, MSPKmerCounter is the slowest. Even if the running time of these tools shows slight difference, all the tools can finish the run in relatively less time (minutes range).

We also tested the tools on four bigger simulated datasets. From Table 3, we can see that KMC2 is the fastest, but for the 2 TB dataset, it runs out of memory. As the dataset increased from 250 GB to 1 TB, the time consumption increased by 11.3 times, and the efficiency is only about 35%. For MSPKmerCounter, it cannot run the simulated dataset larger than 500 GB. Only Jellyfish can handle 2 TB simulated dataset, but it takes more than one day to count all the k-mers. Our proposed k-mer counter is designed for distributed environment with more memories, thus it is tested in the following step with more cores.

4.2 Scaling Test

To evaluate the scalability of the proposed tool, we conducted both the weak scaling and strong scaling tests. The size of simulated datasets ranges from 250 GB to 2 TB, and the number of CPU cores increases correspondingly from 1024 to 8192. The k value is set to 31. Results are shown in Figs. 2 and 3.

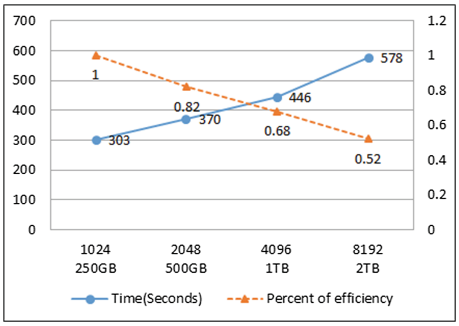

Fig. 2. Execution time and efficiency for weak scaling

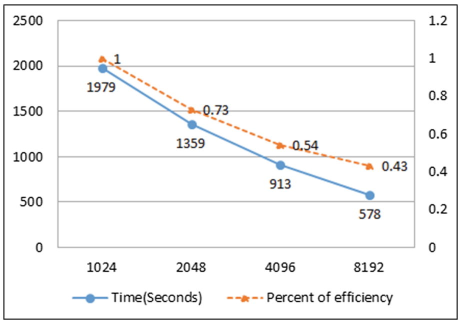

Fig. 3. Execution time and efficiency for strong scaling

Weak Scaling Test: For weak scaling test, we performed an experiment with 4 simulated datasets generated in the previous section. The data size increases from 250G to 2 TB as the number of cores increases from 1024 to 8192. From Fig. 2, we observe that when the cores and the data size doubled, the execution time shows a slightly increase. The ideal situation is a straight line. While the real human sequencing dataset has a fixed distinct k-mer number, and the rising number of distinct k-mers in simulated datasets accounts for this increase. When the number of cores doubles, the rising communication complexity will also affect the performance.

Strong Scaling Test: To analyze the strong scaling performance, we conduct a test on 2 TB simulated dataset with a fix problem size and double the number of cores in each round. Figure 3 shows that the proposed k-mer counter can scale to 8192 cores on 2 TB simulated dataset with more than 200 billion distinct k-mers, the execution time is 578 s. We note that both Bloomfish and kmerind can scale to 1536 cores on 384 GB sequence data from 1000 Genome Data on Comet at San Diego Supercomputer Center. Additionally, Bloomfish can scale to 3072 cores on 24 TB dataset in Tianhe-2A Supercomputer, while the scalability is still lower than our proposed tool. Besides, the strong scaling test results of our proposed tool show a linear decrease in execution time. Similar to the weak scaling test, the communication becomes more complex when the number of cores doubled, and finally a parallel efficiency of 43% for 8192 cores is obtained (1024 cores as a baseline).

5 Conclusions

K-mer counting has become an essential component in bioinformatics, which provides much important information for other bioinformatics pipelines. In this paper, we present a new distributed k-mer counter, which can take advantage of high performance clusters to count k-mers on thousands of cores with limited time. Moreover, it also has a higher scalability than Bloomfish and Kmerind.

Acknowledgements. This work is supported by Guangdong Provincial Department of Science and Technology under grant No. 2016B090918122, National Key Research and Development Program of China under grant No. 2016YFB0201305, the Science Technology and Innovation Committee of Shenzhen under grant No. JCYJ20160331190123578 and No. GJHZ20170314154722613, National Science Foundation of China under grant no. U1435215, 61702494 and 61433012, Shenzhen Basic Research Fund under grant no. JCYJ20170413093358429, Special Program for Applied Research on Super Computation of the NSFC-Guangdong Joint Fund under grant No. U1501501, and Youth Innovation Promotion Association, CAS to Yanjie Wei.

References

1. Meng, J., Wang, B., Wei, Y., Feng, S., Balaji, P.: SWAP-assembler: scalable and efficient genome assembly towards thousands of cores. BMC Bioinform. **15**, S2 (2014)
2. Simpson, J.T., Wong, K., Jackman, S.D., Schein, J.E., Jones, S.J., Birol, I.: Abyss:a parallel assembler for short read sequence data. Genome Res. **19**(6), 1117–1123 (2009)

3. Kelley, D.R., Schatz, M.C., Salzberg, S.L.: Quake: quality-aware detection and correction of sequencing errors. Genome Biol. **11**(11), R116 (2010)
4. Kent, W.J.: Blatthe blast-like alignment tool. Genome Res. **12**(4), 656–664 (2002)
5. Marcais, G., Kingsford, C.: A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. Bioinformatics **27**(6), 764–770 (2011)
6. Deorowicz, S., Kokot, M., Grabowski, S., Debudaj-Grabysz, A.: Kmc 2: fast and resource-frugal k-mer counting. Bioinformatics **31**(10), 1569–1576 (2015)
7. Li, Y., et al.: Mspkmercounter: a fast and memory efficient approach for k-mer counting. arXiv preprint arXiv:1505.06550 (2015)
8. Rizk, G., Lavenier, D., Chikhi, R.: Dsk: k-mer counting with very low memory usage. Bioinformatics **29**(5), 652–653 (2013)
9. Melsted, P., Pritchard, J.K.: Efficient counting of k-mers in dna sequences using a bloom filter. BMC Bioinform. **12**(1), 333 (2011)
10. Roy, R.S., Bhattacharya, D., Schliep, A.: Turtle: identifying frequent k-mers with cache-efficient algorithms. Bioinformatics **30**(14), 1950–1957 (2014)
11. Zhang, Q., Pell, J., Caninokoning, R., Howe, A., Brown, C.T.: These are not the k-mers you are looking for: efficient online k-mer counting using a probabilistic data structure. PLOS ONE **9**(7), e101271 (2014)
12. Pan, T., Flick, P., Jain, C., Liu, Y., Aluru, S.: Kmerind: a flexible parallel library for k-mer indexing of biological sequences on distributed memory systems. IEEE/ACM Trans. Comput. Biol. Bioinform. (2017)
13. Gao, T., Guo, Y., Wei, Y., Wang, B., Lu, Y., Cicotti, P., Balaji, P., Taufer, M.: Bloomfish: a highly scalable distributed k-mer counting framework. In: ICPADS IEEE International Conference on Parallel and Distributed Systems. IEEE (2017). http://www.futurenet.ac.cn/icpads2017/?program-Gid_33.html
14. Gao, T., Guo, Y., Zhang, B., Cicotti, P., Lu, Y., Balaji, P., Taufer, M.: Mimir: Memory-efficient and scalable mapreduce for large supercomputing systems. In: 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1098–1108. IEEE (2017)
15. Meng, J., Seo, S., Balaji, P., Wei, Y., Wang, B., Feng, S.: SWAP-assembler 2: optimization of de novo genome assembler at extreme scale. In: 2016 45th International Conference on Parallel Processing (ICPP), pp. 195–204. IEEE (2016)
16. Georganas, E., Buluc, A., Chapman, J., Hofmeyr, S., Aluru, C., Egan, R., Oliker, L., Rokhsar, D., Yelick, K.: Hipmer: an extreme-scale de novo genome assembler. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, p. 14. ACM (2015)