Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 134 (2018) 219–226

Procedia Computer Science

www.elsevier.com/locate/procedia

The 13th International Conference on Future Networks and Communications (FNC 2018)

Third-generation sequencing data analytics on mobile devices: cache oblivious and out-of-core approaches as a proof-of-concept

Franco Milicchio[a,*], Marco Oliva[a], Christina Boucher[b] and Mattia Prosperi[c]

[a]*Department of Engineering, Roma Tre University, Rome Italy*
[b]*Department of Computer Information Science Engineering, University of Florida, Gainesville, Florida, USA*
[c]*Department of Epidemiology, University of Florida, Gainesville, Florida, USA*

Abstract

Mobile (third-generation) sequencing technologies, including Oxford Nanopore's MinION and SmidgION, have the benefit of outputting long sequence reads (up to hundred thousands of bases) in a portable manner. These sequencing devices fit in the palm of a hand and only require a USB outlet. Unfortunately, the development of data analysis tools for these technologies is in a nascent stage, impeding on the portability of these devices. The objective of this work is to introduce an out-of-core approach to port Nanopore analytics on mobile devices such as tablets or smartphones, often used in extreme experimental settings with special ergonomics needs and ease of sterilization. In this paper, we present a serial *k*-mer parser/counter for FAST5 files, and a de Bruijn graph construction method which can run on a hand-held device. In order to accomplish this portability we develop novel cache oblivious data structures and out-of-core chunked processing methods. Our toolset, which we refer to as Nanopore Portable Analytics Library (NanoPAL), wase implemented in ISO C++ v.14 and compiled for Android devices. Using MinION data (*Zaire Ebolavirus* species and others), we evaluate the time required to parse and build the de Bruijn graph with respect to the file sizes and RAM allocation. These metrics were compared to those of minimap/miniasm. On an LG Nexus 5 with 2GB or RAM, 2MB L2 cache and 16GB storage, the out-of-core NanoPAL is able to process FAST5 files at about 30 minutes per 0.5 GB, creating sorted *k*-mer and de Bruijn graph files. The recompiled minimap/miniasm tool cannot complete FAST5 files larger than 170MB. In conjunction with base calling/error correction, and with addition of assembly procedures downstream, NanoPAL can be effectively used to perform analyses of MinION/SmidgION data locally on a mobile device.

Peer-review under responsibility of the scientific committee of the 13th International Conference on Future Networks and Communications, FNC-2018 and the 15th International Conference on Mobile Systems and Pervasive Computing, MobiSPC 2018.

Keywords: Next-generation sequencing; minION; de novo assembly; de Bruijn graph; *k*-mer; cache oblivious; out-of-core.

* Corresponding author. Tel.: +39-06-5733-3201; fax: +39-06-5733-6444.
E-mail address: franco.milicchio@uniroma3.it

Peer-review under responsibility of the scientific committee of the 13th International Conference on Future Networks and Communications, FNC-2018 and the 15th International Conference on Mobile Systems and Pervasive Computing, MobiSPC 2018.
10.1016/j.procs.2018.07.164

1. Introduction

In the past decade, next-generation sequencing (NGS) technologies (e.g. Illumina's HiSeq, Pacific Biosciences' PacBio) enabled the generation of genomic and transcripomic data in a cost-effective, high-throughput manner. This has enabled the application of this data to a wide-range of problems stemming from biological and health sciences. Currently genomics, transcriptomics, metabolomics and other –omics sciences are linchpins of basic and translational biomedical research. After high-throughput, miniaturization and portability is the next scientific new leap. Mobile and portable (third-generation) sequencing technologies—e.g. Oxford Nanopore's MinION, SmidgION and VolTRAX for library preparation—literally bring genomics to the palm of a hand (the MinION weighs 90g and measures 10×3×2cm), transforming again biomedical sciences [1]. Nanopore combines long sequence read length (up to hundred thousands of bases) with extreme portability [2,3]. Proof-of-concept studies showed MinION can be used to detect pathogens, e.g. Ebola or tuberculosis, with rapid turnaround time [4-6]—making them well-suited for field applications, such as food safety monitoring in farms and water purification testing.

While MinION and SmidgION need only a USB outlet, the data analysis phases are bound to an available Internet connection and ad hoc computing resources. However, tablets are preferable to desktop and laptop computers in many experimental settings because of their resilience to humidity or spills, ease of sterilization, usability (apps) and ergonomics (touchpad rather than a keyboard). This absence of smartphone/tablet-tailored analytics tools hampers portability, especially if the technology is used in resource-limited settings or remote areas with limited connectivity, which limits the applications to which these technologies were developed for.

To date, a number of commercial and open source tools for Nanopore analytics are available—both cloud-based and local—but there is no software purposely built to run on portable devices such as tablets or smartphones. As one of the few commercial options, Metrichor (https://metrichor.com) provides all-purpose analytics services, from error correction to genome assembly. Among the open-source software: poRe and poretools provide file parsing and visualization [7,8]; Nanocall does base calling [9]; PoreSeq performs de novo/reference error correction and variant calling [10]; Canu and minimap/miniasm provide de novo assembly [11,12].

Majority of de novo assembly methods look at overlaps among sequence reads to reconstruct the genome, using either the overlap-layout-consensus (OLC) or the de Bruijn graph [13]. In the ideal case, with the OLC, a read is a node, an overlap between two reads is an arc, and the genome is assembled by finding a Hamiltonian path that visits all reads once (NP-complete problem). In the de Bruijn, a read is broken down into smaller strings of fixed length k (k-mers), each k-mer is an arc connecting two overlapping (k-1)-mers, and the genome is assembled by finding an Euler path that visits all arcs once (linear time complexity). In practice, there are graph reduction/transformation, heuristic assembly of consistent regions, unitigs, and then pre- and post-scaffolding guided by multiple alignment [14].

There are several challenges to port Nanopore analytics on to mobile devices, including architectural differences, memory-computational constraints, and power/heating limitations. To address these challenges, we present novel, cache-oblivious, out-of-core approach to tackling the first two analysis challenges: k-mer counting and de Bruijn graph construction. Hence, we first show how to efficiently parse MinION's FAST5 file using an HDF5 library for Android OS, then we introduce cache oblivious k-mer and de Bruijn data structures (which are the basis of most second- and third-generation sequencing analytics algorithms) and lastly, present their out-of-core implementation. We compare the results of our methods with the Android recompiled versions of minimap/miniasm. Of note, minimap/miniasm performs steps of an OLC assembly, so it is only in part comparable to our method, but it could be compiled seamlessly on Android, differently from others. The proposed framework is programmed using ISO C++ v.14 and takes the collective name of Nanopore Portable Analytics Library (NanoPAL). NanoPAL is available under the BSD-3 license at https://fmilicchio.bitbucket.io/nanopal.html.

2. Methods

2.1. Out-of-core approach: conceptual framework

Even if smartphones and tablets have RAM and cache comparable to that or desktop/laptop computers, the usage is often limited by the operating system (both iOS and Android). Analysis of high-throughput sequencing data often requires a considerable amount of memory and CPU resources, even when compressed data structures are used [15].

An out-of-core approach that uses paged virtual memory is a feasible solution to the memory constraint problems only if the decrease in performance due to I/O and physical memory accesses is acceptable. To overcome this challenge, we propose a cache oblivious out-of-core approach, to build a k-mer spectrum and de Bruijn graph from FAST5 files, minimizing accesses to physical memory through chunked file processing and sorting. In the following sections, we present the details of this approach.

2.2. File parsing

In order to parse FAST5 files, we used the HDF5 standard library (www.hdfgroup.org) on which the FAST5 format is based. We previously theorized cache oblivious k-mer data structures, with an in house file parser for FASTA and FASTQ file formats for iOS and Android [16,17], which we extend here to build the k-mer graph. At the moment our procedures ignore quality score, supposing previously cleaned data. For high-noisy data such MinION, the assumption is realistic only upon a previously base-called and quality-trimmed raw sequence file, and either an independent error correction module such as LoRMA or LoRDEC [18,19], or standing an assembly tool with error correction, e.g. as in the string-graph SGA or in the de Bruijn graph SPAdes assembler [20,21].

2.3. Counting and storing k-mers

Even though many efficient data structures for counting and storing k-mers are available [22-24], there are scarce approaches for out-of-core processing [25]. Thus, we implement a simple and minimalist representation of k-mers, based on a bit-wise representation of the four ACTG nucleotides, and bit-to-bit operations to verify string overlaps and reverse-complement equivalency, as previously described [26]. We note this representation can be replaced by a different one as we use a generic template programming approach, like SeqAn [27,28].

To allow feasibility of processing of Gigabyte-sized sequence data on a mobile device with limited amount of usable RAM (usually a few hundred MB over the total) and cache, we devised the out-of-core approach constituted by a cache oblivious data structure together with a chunked file processing.

The k-mer spectrum creation is done in two phases. In the first phase, the FAST5/FASTQ/FASTA file is parsed and reads are broken down in k-mers that are stored, along with their counts, in a cache oblivious vector (which has less overhead then a hashmap) whose size is equal to the available RAM. When it is full, this is sorted and then printed into a file in physical memory. Once the vector has been emptied parsing can continue. For each block, k-mers with partial counts are therefore, written.

After this phase, a second phase starts and k-mers in each block are merged using an out-of-core merge sort algorithm which allows an arbitrary allocation of spatial resources. **Fig. 1** illustrates the steps of these two phases.

As a clean-up, after ordering, all *k*-mers with frequency of one are purged (this step is optional and customizable on any desired count).

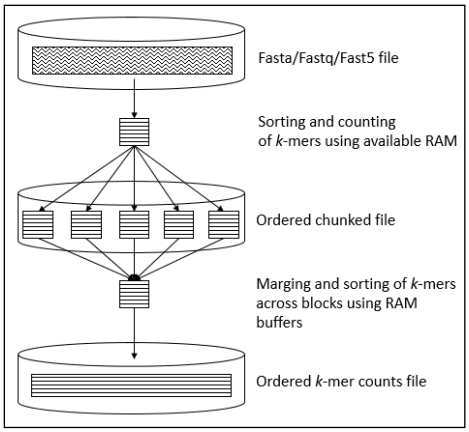

Fig. 1. Out-of-core *k*-mer merging and sorting. Blocks created during the parsing are merged and sorted using an out-of-core merge sort with user-defined space allocation, via file chunking and RAM buffering.

2.4. De Bruijn graph building

The de Bruijn graph can be defined constructively as follows: all unique k-length sequences (k-mers) are found in the set of input reads, a directed edge of each k-mer is built, and the nodes of the edge are labelled as the prefix and suffix made by (k-1)-mers of the corresponding k-mer, gluing nodes with the same label together.

The de Bruijn graph is constructed from the k-mer spectrum and overlap information using an adjacency list implemented through an out-of-core array with divisions into blocks. Default search and insertion operators are overridden and rewritten accordingly. The RAM is used as a cache, which guarantees better performance as compared to classical paging. This ad hoc array solution was found more performant than a vector for files of 1GB or larger. The out-of-core array contains the k-mer and four integers with the indices of the overlapping k-mers, as shown in **Fig. 2.**

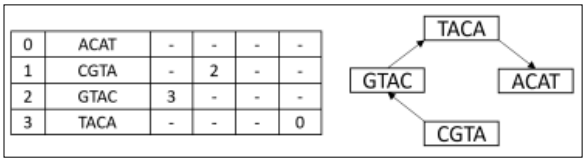

Fig. 2. Out-of-core de Bruijn graph. The out-of-core array contains the *k*-mer and indices of the four possible overlapping *k*-mers.

3. Results

To evaluate the NanoPAL, an LG Nexus 5 device (2.26 GHz quad-core Snapdragon 800, 2GB RAM, 4 KiB + 4 KiB L0 cache, 16 KiB + 16 KiB L1 cache and 2 MiB L2 cache, 16GB internal storage) mounting Android OS v.6.0.1 with api-level 23, was used. Three public MinION FAST5 sequencing experiments were downloaded from Genbank's sequence read archive and used as test datasets: (i) Oooceraea biroi (accession no. SRX3429051); (ii) Zaire ebolavirus species (accession no. ERR1248114); and (iii) Homo sapiens metagenomics sequencing of

prosthetic joint infection (accession no. ERR2195910). On the same files, we run the recompiled version of minimap/miniasm, and compared processing times with NanoPAL using both full file sizes and ad hoc sampling.

3.1. File parsing, k-mer processing, and graph building

We evaluated the OOC performance of NanoPAL first by varying the available RAM (50MB, 100MB, and 200MB), and then by varying the *k*-mer spectrum composition using the different real-world data sets.

Given the latency due to I/O, small variations in the allocated RAM do not affect sensibly the execution times. However, shorter times are observed when the RAM is increased (see **Fig. 3**).

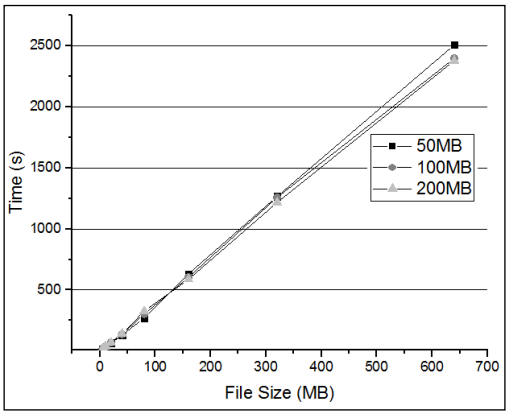

Fig. 3. NanoPAL performance by RAM allocation. Execution times in relation to RAM allocation for the out-of-core *k*-mer spectrum and de Bruijn graph construction, using FAST5 samples of 100-700 MB (*Spodoptera frugiperda* species dataset, accession no. SRR6042592).

We then fixed the RAM to 200MB and executed tests for the three different FAST5 retrieved from Genbank. **Table 1** summarizes results in relation to input file size and *k*-mer spectrum size: the size of the out-of-core file is given, and execution times are broken down by: sorted k-mer file size (F_1); cleaned de Bruijn graph file size (F_2); and time to build de Bruijn graph (T_2).

The total time increases linearly with the file size, and on average every half GB is processed in about half an hour to create both the sorted *k*-mer file and the de Bruijn graph.

Table 1. Out-of-core *k*-mer counting and graph construction.

Dataset	File size	F_1	F_2	T_2	Tot time
i	400MB	2289MB	59MB	2m	32m
i	600MB	3425MB	104MB	5m	50m
ii	100MB	572MB	4MB	8s	7m
ii	150MB	858MB	8MB	17s	10m
iii	200MB	733MB	114MB	1m	14m
iii	400MB	1417MB	244MB	3m	31m
iii	600MB	2082MB	376MB	5m	48m
iii	1GB	2760MB	507MB	12m	1h5m
iii	2GB	3435MB	634MB	37m	2h38m

F_1: sorted *k*-mer file size (all *k*-mers); F_2 cleaned de Bruijn graph file size; T_2 time to build de Bruijn graph; i: *O. biroi* species; ii: *Zaire Ebolavirus*; iii: *H. sapiens* metagenomics.

3.2. Comparison with minimap/miniasm

We compiled our code and recompiled minimap/miniasm using the Android SDK/NDK (https://developer.android.com/ndk/index.html), C++ v.14, with minimal modifications made to the minimap/miniasm source code.

Using the FAST5 data sets i-iii, we created various file samples of increasing size between 50 MB and 2GB. We run minimap/miniasm on each file and the program completed successfully for all files up to 170MB, then halted with error. **Fig. 4** compares minimap/miniasm with NanoPAL, which completed successfully all files. On average, out-of-core NanoPAL was 12x slower than minimap/miniasm.

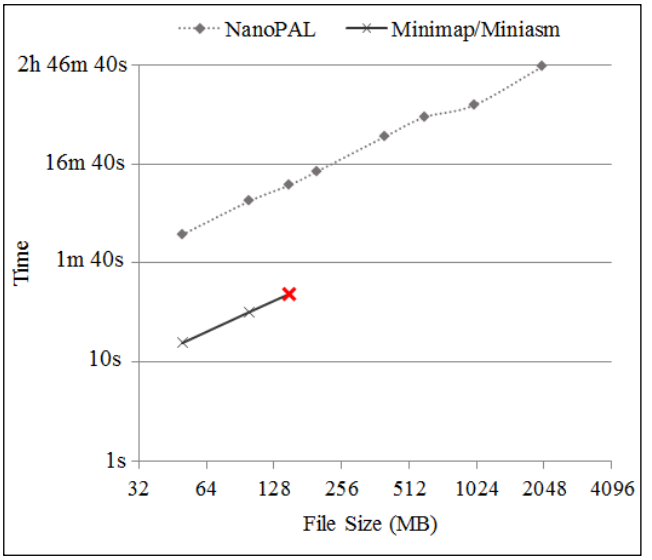

Fig. 4. NanoPAL vs. minimap/miniasm. Execution times of NanoPAL and minimap/miniasm as a function of the input FAST5 file size (average over all datasets): minimap/miniasm halts at 170 MB, whilst the out-of-core NanoPAL completes all files.

4. Discussion

We presented NanoPAL, an open source, generic programming template library designed to enable portable analysis of third-generation sequencing data, including Oxford's Nanopore MinION and SmidgION.

NanoPAL features a FAST5 file parser and cache oblivious, out-of-core methods for building the *k*-mer spectrum and the de Bruijn graph. Even with the usage of non-compressed data structures, NanoPAL offers a reasonable memory-runtime tradeoff, which opens up a vast perspective for analyzing real experimental data –from a few to several GB– directly on mobile devices mounting Android or iOS.

Only a few *k*-mer spectrum builders that exploit external memories are available [29,30], and a tool for string graph build-up [31]. Overall, efficient representations of de Bruijn graphs in external memory are scarce, but such structures are vital for enabling truly portable analytics of third-generation sequencing data.

4.1. Limitations

This work has some limitations. First, the *k*-mer and the de Bruijn graph data structures could be optimized, e.g. using compression, allowing a smaller memory footprint and executing more operations in the fast memory hierarchies. Second, memory constraints of the operating systems on mobile devices often do not practically permit usage of all available RAM and cache levels. Third, we have not evaluated battery consumption and ignored device

heating issues. Fourth, NanoPAL does not provide a base calling method, and one must rely on other preprocessing tools, such as Nanocall [32]. Finally, at the moment NanoPAL does not feature a de novo genome assembly algorithm, although it is possible to test other software via generation of standard FASTG files (http://fastg.sourceforge.net/).

4.2. Perspectives

As NanoPAL uses the generic template approach, we foresee the improvement of its methods. For instance, a cache-efficient Bloom filter [33] could be employed for the *k*-mer spectrum. Also, usage of minimum substring partitioning for out-of-core [34] can improve performance. Another possible improvement for the de Bruijn graphs is through usage of succinct (compressed): an external memory version of the Ferragina-Manzini index has been proposed [35], and very recently the first implementation of suffix trees that avoids direct string comparisons has been released [36].

Acknowledgements

This work has been supported by the "Virogenesis" project which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 634650.

References

[1] Pennisi, E. Pocket-sized sequencers start to pay off big. *Science* **356**, 572-573 (2017).
[2] Jain, M., Olsen, H.E., Paten, B. & Akeson, M. The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community. *Genome Biol* **17**, 239 (2016).
[3] Mikheyev, A.S. & Tin, M.M. A first look at the Oxford Nanopore MinION sequencer. *Molecular ecology resources* **14**, 1097-1102 (2014).
[4] Quick, J.*, et al.* Real-time, portable genome sequencing for Ebola surveillance. *Nature* **530**, 228-232 (2016).
[5] Votintseva, A.A.*, et al.* Same-Day Diagnostic and Surveillance Data for Tuberculosis via Whole-Genome Sequencing of Direct Respiratory Samples. *Journal of clinical microbiology* **55**, 1285-1298 (2017).
[6] Lemon, J.K., Khil, P.P., Frank, K.M. & Dekker, J.P. Rapid Nanopore Sequencing of Plasmids and Resistance Gene Detection in Clinical Isolates. *Journal of clinical microbiology* (2017).
[7] Watson, M.*, et al.* poRe: an R package for the visualization and analysis of nanopore sequencing data. *Bioinformatics* **31**, 114-115 (2015).
[8] Loman, N.J. & Quinlan, A.R. Poretools: a toolkit for analyzing nanopore sequence data. *Bioinformatics* **30**, 3399-3401 (2014).
[9] David, M., Dursi, L.J., Yao, D., Boutros, P.C. & Simpson, J.T. Nanocall: an open source basecaller for Oxford Nanopore sequencing data. *Bioinformatics* **33**, 49-55 (2017).
[10] Szalay, T. & Golovchenko, J.A. De novo sequencing and variant calling with nanopores using PoreSeq. *Nature biotechnology* **33**, 1087-1091 (2015).
[11] Koren, S.*, et al.* Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome research* **27**, 722-736 (2017).
[12] Li, H. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics* **32**, 2103-2110 (2016).
[13] de Laat, M.A., Sillence, M.N., McGowan, C.M. & Pollitt, C.C. Continuous intravenous infusion of glucose induces endogenous hyperinsulinaemia and lamellar histopathology in Standardbred horses. *Veterinary journal* **191**, 317-322 (2012).
[14] Simpson, J.T. & Pop, M. The Theory and Practice of Genome Sequence Assembly. *Annual review of genomics and human genetics* **16**, 153-172 (2015).
[15] Deorowicz, S. & Grabowski, S. Data compression for sequencing data. *Algorithms Mol Biol* **8**, 25 (2013).
[16] Milicchio, F. & Prosperi, M. Efficient data structures for mobile de novo genome assembly by third-generation sequencing. *Procedia Computer Science* **110**, 440-447 (2017).
[17] Milicchio, F., Tradigo, G., Veltri, P. & Prosperi, M. High-performance data structures for de novo assembly of genomes: cache oblivious generic programming. in *Proceedings of the 7th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics* 657-662 (ACM, Seattle, WA, USA, 2016).
[18] Salmela, L., Walve, R., Rivals, E. & Ukkonen, E. Accurate self-correction of errors in long reads using de Bruijn graphs. *Bioinformatics* **33**, 799-806 (2017).
[19] Salmela, L. & Rivals, E. LoRDEC: accurate and efficient long read error correction. *Bioinformatics* **30**, 3506-3514 (2014).

[20] Simpson, J.T. & Durbin, R. Efficient de novo assembly of large genomes using compressed data structures. *Genome Research* **22**, 549-556 (2012).
[21] Bankevich, A.*, et al.* SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing. *Journal of Computational Biology* **19**, 455-477 (2012).
[22] Kowalski, T., Grabowski, S. & Deorowicz, S. Indexing Arbitrary-Length k-Mers in Sequencing Reads. *PLoS One* **10**(2015).
[23] Melsted, P. & Pritchard, J.K. Efficient counting of k-mers in DNA sequences using a bloom filter. *BMC Bioinformatics* **12**(2011).
[24] Zhang, Q., Pell, J., Canino-Koning, R., Howe, A.C. & 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**, e101271 (2014).
[25] Rizk, G., Lavenier, D. & Chikhi, R. DSK: k-mer counting with very low memory usage. *Bioinformatics* **29**, 652-653 (2013).
[26] Milicchio, F., Buchan, I.E. & Prosperi, M.C.F. A* fast and scalable high-throughput sequencing data error correction via oligomers. in *2016 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)* 1-9 (2016).
[27] Milicchio, F., Rose, R., Bian, J., Min, J. & Prosperi, M. Visual programming for next-generation sequencing data analytics. *BioData Min* **9**, 16 (2016).
[28] Doring, A., Weese, D., Rausch, T. & Reinert, K. SeqAn an efficient, generic C++ library for sequence analysis. *BMC Bioinformatics* **9**, 11 (2008).
[29] Erbert, M., Rechner, S. & Muller-Hannemann, M. Gerbil: a fast and memory-efficient k-mer counter with GPU-support. *Algorithms Mol Biol* **12**, 9 (2017).
[30] Janin, L., Schulz-Trieglaff, O. & Cox, A.J. BEETL-fastq: a searchable compressed archive for DNA reads. *Bioinformatics* **30**, 2796-2801 (2014).
[31] Bonizzoni, P., Vedova, G.D., Pirola, Y., Previtali, M. & Rizzi, R. LSG: An External-Memory Tool to Compute String Graphs for Next-Generation Sequencing Data Assembly. *J Comput Biol* **23**, 137-149 (2016).
[32] David, M., Dursi, L.J., Yao, D.L., Boutros, P.C. & Simpson, J.T. Nanocall: an open source basecaller for Oxford Nanopore sequencing data. *Bioinformatics* **33**, 49-55 (2017).
[33] Roy, R.S., Bhattacharya, D. & Schliep, A. Turtle: identifying frequent k-mers with cache-efficient algorithms. *Bioinformatics* **30**, 1950-1957 (2014).
[34] Li, Y.*, et al.* Memory efficient minimum substring partitioning. *Proc. VLDB Endow.* **6**, 169-180 (2013).
[35] Ferragina, P., Gagie, T. & Manzini, G. Lightweight Data Indexing and Compression in External Memory. *Algorithmica* **63**, 707-730 (2012).
[36] Louza, F.A., Telles, G.P., Hoffmann, S. & Ciferri, C.D.A. Generalized enhanced suffix array construction in external memory. *Algorithm Mol Biol* **12**(2017).