8-2015

A Parallel Algorithm for Compression of Big Next-Generation Sequencing Datasets

Sandino N. Vargas Perez
Western Michigan University

Fahad Saeed
Western Michigan University, fahad.saeed@wmich.edu

A Parallel Algorithm for Compression of Big Next-Generation Sequencing (NGS) Datasets

Sandino Vargas Pérez
Computer Science Department
Western Michigan University
Kalamazoo MI 49008 USA
Email: sandinonarciso.vargasperez@wmich.edu

Fahad Saeed*
Department of Electrical and Computer Engineering
Western Michigan University
Kalamazoo MI 49008-5329 USA
Email: fahad.saeed@wmich.edu
*To whom correspondence should be addressed

***Abstract*—With the advent of high-throughput next-generation sequencing (NGS) techniques, the amount of data being generated represents challenges including storage, analysis and transport of huge datasets. One solution to storage and transmission of data is compression using specialized compression algorithms. However, these specialized algorithms suffer from poor scalability with increasing size of the datasets and best available solutions can take hours to compress Gigabytes of data. In this paper we introduce paraDSRC, a parallel implementation of DSRC using a message passing model that presents reduction of the compression time complexity by a factor of $O(\frac{1}{p})$. Our experimental results show that paraDSRC achieves compression times that are 43% to 99% faster than DSRC and compression throughputs of up to 8.4GB/s on a moderate size cluster. For many of the datasets used in our experiments super-linear speedups have been registered, making the implementation strongly scalable. We also show that paraDSRC is more than 25.6x faster than comparable parallel compression algorithms. The code will be available in author's website if paper is accepted.**

I. Introduction

Next-generation sequencing (NGS) technologies can generate enormous amount of data, delivering outputs of up to 1 terabase[1] of genetic sequence data in a single run [1]. These large datasets require significant time to be processed and transformed into relevant information [2]. Keeping these NGS datasets intact before the processing stage is important due to the preservation of key elements that may become meaningful in future analyses. Storing and transferring these massive amounts of data (generated very fast and cheaply) represents a challenge that has led to creation of highly specialized compression techniques such as Quip [3], G-SQZ [4], DSRC [5], KungFQ [6], SeqDB [7], SOLiDzipper [8]. These algorithms take into account the characteristics of the FASTQ format [9] in which the DNA sequence read and its corresponding per base quality score are structured and stored.

Any compression algorithm loses its usefulness if it takes more time to compress the data than it would take a scientist to analyze it. In order to speedup the process, general purpose compression algorithms have been parallelized using shared memory models such as pigz[2] (pronounced "pig-zee", a parallelization of gzip) which exploits multicore technologies and uses the pthread library and PBZIP2[3] a parallel version of bzip2 that also uses pthreads. A message passing version of bzip2 named MPIBZIP2[4] is also available. Parallel compression algorithms specialized in NGS datasets are scarce. The authors are only aware of one parallel algorithm specific to NGS data, named DSRC 2 [10]. DSRC 2 is an industry oriented, multi-threaded parallel version of DSRC with faster compression times, variable throughput and comparable compression ratios to existing solutions.

In this paper, we present the first algorithm that uses a distributed memory-architecture to parallelize the compression process specific to NGS data. As a proof-of-concept we use DSRC as the algorithm underlying our parallelization strategy because it has the best overall performance in terms of speed and compression ratio [11]. Our experiments show that by using a message passing model on a distributed memory-architecture and load balancing the reading/writing phases among the working processes, faster compression times and higher throughput can be achieved, making the algorithm scalable with increasing number of processing units and datasets.

In the following sections we analyze the design of paraDSRC and evaluate the performance obtained by running the algorithm with small to considerably big (∼10MB to ∼1TB) FASTQ input files.

II. Background Information

FASTQ is the most commonly used NGS dataset format [9]. As shown in Fig. 1 it contains many records, each of which is composed by four lines: (i) a title and optional description line, (ii) DNA sequence read line, (iii) a plus '+' sign followed by optional repetition of the title line and (iv) the quality score line . DSRC processes the FASTQ file by dividing it into three streams of data:

1) *Title information*: reads, analyzes and compresses the title stream using different techniques such as compact encoding, delta or differential encoding for numeric portions of the title, entropy coding and Huffman encoding.
2) *DNA sequence*: this stream is compressed using LZ77-style encoding. The plus sign (on the 3rd line of a record) is only use as an "end of DNA sequence" marker.

[1]Equivalent to 10^{12} base pairs
[2]http://zlib.net/pigz/
[3]http://compression.ca/pbzip2/
[4]http://compression.ca/mpibzip2/

3) *Quality score*: this stream is encoded depending on its characteristics; if the quality score is quasi random it uses order-0 Huffman coding, otherwise, for repetitive sequences it uses Run-Length encoding (RLE).

```
@ERR005195.1 BGI-FC30BFTAAXX_5_1_000:1689/2
CTCCCATATCCTTAGAGAAAATCCCCAATGCCTAGT
+
IIIIIIIIIIIIIIIIIIIIIIIIII'8=;I?DG8
@ERR005195.2 BGI-FC30BFTAAXX_5_1_000:125/2
TGTTTGGCAAGGTCCTACAAAAGTTGCAACTCTCAC
+
IIIIIIIIIIIIIIIIIII2?9IIII*7)IIII'-
```

Fig. 1. FASTQ file containing 2 records. Each record's title starts with the ASCII character '@' followed by a description. The second line contains the DNA sequence read. The ASCII character '+' is used in the third line to notify the end of the DNA sequence (an optional repetition of the title is sometimes included in this line). The fourth line of the record is the quality score and it's always the same length of the second line.

According to [11] a big portion (43.80%) of the total compression time of DSRC is spent in analyzing the title portion of a FASTQ record, but further profiling made us conclude that this algorithm tends to be I/O bounded as the dataset size increases. Therefore, the design of our parallel strategy takes this into account along with other factors for scalability.

III. Proposed Parallel Strategy

For our parallel implementation, a FASTQ file of size N is partitioned equally among p processes. This division will create a working region of size $\frac{N}{p}$ for each p_i, $\forall i \in \mathbb{N} : i$ from 0 to $p-1$, as shown in Fig. 2. Due to the nature of the FASTQ file and the imposed arbitrary division, p_i cannot know if a complete record (containing the 4 lines explained in the previous section) is present at the start and/or at the end of its working region. For this reason we use a small overlap between them to guarantee that each p_i's working region contains complete records.

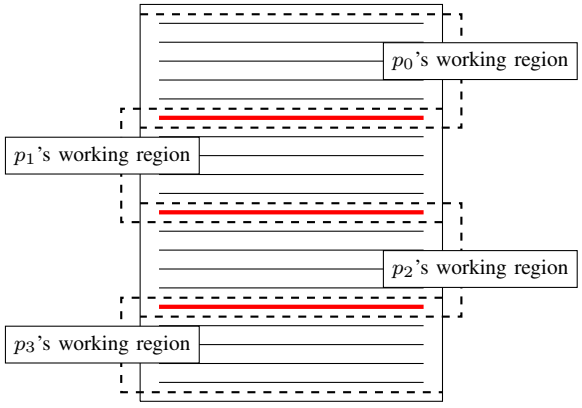

Fig. 2. FASTQ file partition between 4 processes into working regions of size $\frac{N}{p}$ each. The thick red lines represent an overlap between processes' working regions.

After working regions are assigned, each process identifies the start of the first complete record at the beginning of its region. p_i will continue fetching records until it reaches the end of its allocated region. If it doesn't have a complete record after reading the last byte, it will continue through the small overlap portion o and stop when the first full record is stored. This will ensure that p_i will finish reading its last record at byte b since $p_{(i+1)}$ started reading its first record at byte $b+1$.

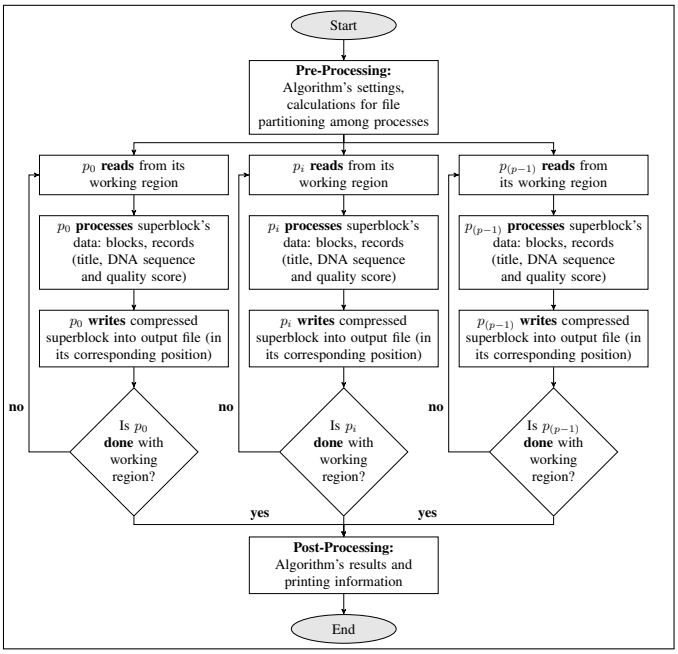

Fig. 3. Flowchart of the design for p processes

Each process will then analyse and compress the title, DNA and quality score portion of the stored record. After enough records are processed each p_i will write chunks of k bytes of compressed data in a non-blocking, non-collective fashion using a shared file pointer to the output file. Fig. 3 shows the flowchart for paraDSRC using MPI.

The algorithm will terminate when every p_i has processed the last full record of their working regions and any remaining data is compressed and written. The steps taken to compress a FASTQ file with paraDSRC are illustrated using pseudo-code in Fig. 4.

A. Analysis of Computation and Communication Costs of the Proposed Parallel Strategy

Overall, DSRC has a linear asymptotic growth of $O(N)$, dictated mostly by the processing of the title information and the reading (and preparation) of the records to be compressed. We refer to N as the size of the problem given in number of records. In our parallel solution, each processor p_i will get an approximately equal amount of records $w = \frac{N}{p}$.

Every parallel implementation incurs in some amount of communication overhead and ours is no exception [12]. In order to make the overhead costs T_o smaller, we reduce the communication rounds to only 2 phases; reading the input FASTQ file and writing the compressed data to the output.

For the reading phase, let us assume l is the size (given as number of records) of a chunk of data that a processor p_i will

Require: p_i processing units ($\forall i \geq 2$) and FASTQ file $input.fastq$
Ensure: $output.dsrc$ file size $0 < M < N$ and all p exit correctly

1: **procedure** PARADSRC(p, $input.fastq$)
2: $l \leftarrow 4\ megabytes$ // bytes to be read from $pWorkingRegion$
3: $o \leftarrow 500\ bytes$ // overlap
4: $k \leftarrow 8\ megabytes$ // bytes of compressed data to be written
5: N, M // N is the size of input, M size of output file
6: $pWorkingRegion \leftarrow N/p$
7: $regionStart$, $regionEnd$
8: $output.dsrc$
9: call of **mpi_init()** // initiate MPI environment
10: p_i finds byte b where the first complete record in its $pWorkingRegion$ starts.
11: $regionStart \leftarrow b$
12: **while** $regionEnd$ **is not** reached **do**
13: p_i **reads** $l + o$ bytes **from** $pWorkingRegion$ **at** $regionStart$
14: **repeat**
15: p_i **inserts** $title$ **into** Rec
16: p_i **inserts** DNA **into** Rec
17: p_i **inserts** $QualityScore$ **into** Rec
18: $bytesProcessed \leftarrow$ total size of data inserted
19: $CompressData \leftarrow$ **compress**(Rec)
20: **if** size of $CompressData \geq k$ **then**
21: p_i **writes** k bytes of $CompressData$ **to** $output.dsrc$
22: **end if**
23: **until** complete Rec is processed **and** $bytesProcessed \geq l$
24: $regionStart \leftarrow regionStart + bytesProcessed$
25: **end while**
26: **if size**($CompressData$) > 0 **then**
27: p_i **writes** remaining bytes of $CompressData$ **to** $output.dsrc$
28: **end if**
29: p_i calls **mpi_finalize()**
30: **end procedure**

Fig. 4. Pseudo-code for paraDSRC using p processes and FASTQ file $input.fastq$

read out of N. There will be $\frac{N}{l}$ chunks to be read in total. Since the reading occurs simultaneously, the communication overhead will have a complexity of $O(\frac{N}{lp})$.

To analyze the communication overhead incurred in the writing phase, we define r as the compression ratio of the algorithm given by $r = \frac{N}{M}$, where M is the size of the resulting compressed data (DSRC file). Let's call k the size of compressed data chunk that every processor will write as they become available. Approximately $\frac{M}{k}$ chunks will be written by p processors, hence the complexity for the second phase will be represented by $O(\frac{N}{rkp})$. The total time complexity of the parallel execution given by $T_P = T_c + T_o$, will be:

$$Computation\ cost,\ T_c = O\left(\frac{N}{p}\right) \tag{1}$$

$$Communication\ cost,\ T_o = O\left(\frac{N}{lp}\right) + O\left(\frac{N}{rkp}\right) \tag{2}$$

$$T_P \approx O\left(\frac{N}{p}\right) + O\left(\frac{N}{lp}\right) + O\left(\frac{N}{rkp}\right) \tag{3}$$

B. Scalability of the Proposed Parallel Strategy

Calculating the speedup S with increasing number of processes and data size will tell us if our strategy has good scalability, i.e., the processing speed will double by doubling the number of processors. We will obtain this ideal ratio or linear Speedup if $S = p$. We will use the definitions and assumptions of [13]: $S = \frac{T_S}{T_P}$, where T_S is the sequential execution time. Substituting T_P with (3) we will have:

$$S = \frac{N}{\frac{N}{p} + \frac{N}{lp} + \frac{N}{rkp}} = \frac{N}{\frac{N}{p}(1 + \frac{1}{l} + \frac{1}{rk})} \tag{4}$$

Assuming that $C = (1 + \frac{1}{l} + \frac{1}{rk})$ is a constant that depends on the implementation parameters, then we can reduce equation (4) to:

$$S = \frac{N}{\frac{NC}{p}} = \frac{p}{C} \tag{5}$$

We know that C will have a value of $1 \geq C \leq 1.2$, so we can re-write equation (4) to have:

$$S \approx p$$

The approximation above indicates that theoretically our parallel implementation should exhibit linear speedup. Continuing with [10] the efficiency $E = \frac{S}{p}$ will be approximately $\Theta(1)$.

IV. RESULTS

To test our implementation we used the Thor compute cluster[5], a high performance parallel computing cluster at Western Michigan University. Thor has 22 compute nodes equipped for different parallel models including 16 nodes for MPI parallel programming. These nodes have a dual Sandy-Bridge Intel Xeon E5-2670 2.6GHz processors (16 cores each) with 128GB of RAM and a 1TB hard drive for temporary work. The cluster uses an ethernet management and a low-latency Infiniband communication network. All nodes have access to network mounted directories on a high performance, high redundancy RAID file server. The cluster implements TORQUE resource manager [14] which is used to specify the resource requirements for the algorithm. OpenMPI version 1.7.3 and GCC version 4.7.3 were used for compilation and execution.

The NGS datasets for the experiment were obtained from the 1000 Genomes Project[6] and are shown in Table I. The column "Renamed" is used to identify the datasets with relation to their sizes (rounded to the closest power of 10), e.g., dataset with ID ERR009075 and size of 1,015 megabytes is renamed "1GB". This identification will be used throughout the section instead of the datasets' IDs.

A. Performance Evaluation

We executed paraDSRC using 2, 4, 8, 16, 32, 64 and 128 processing units with increasing size of the datasets. As can be seen in Fig. 5 the compression time decreases sharply as more processing units are added with an increasing input size.

[5]https://www.cs.wmich.edu/ hpcs

[6]http://www.1000genomes.org

[7]This dataset was constructed by appending ERR022729 multiple times and changing the ID in the title portion of every record.

TABLE I
DATASETS USED FOR THE TEST

IDs	Organism	Platform	Read count	Size(MB)	Renamed
ERR005195	Homo sapiens	ILLUMINA	76,167	9.3	10MB
ERR229788	Homo sapiens	ILLUMINA	372,983	97	100MB
ERR009075	Homo sapiens	ILLUMINA	8,261,260	1,015	1GB
ERR260401	Homo sapiens	ILLUMINA	43,598,329	11,000	10GB
ERR022729	Homo sapiens	ABI_SOLID	468,457,226	104,000	100GB
1T-MIXED[7]	Homo sapiens	ABI_SOLID	1,566,932,640	1,100,000	1TB

TABLE II
PERCENTAGE DECREASE IN COMPRESSION TIME FOR $paraDSRC$

Datasets	Number of Processing Units						
	2	4	8	16	32	64	128
10M	34.06%	71.38%	84.42%	89.76%	92.67%	92.47%	95.26%
100M	56.35%	76.81%	88.23%	93.43%	96.60%	98.11%	98.88%
1G	46.02%	76.74%	88.17%	94.08%	97.05%	98.47%	99.21%
10G	45.42%	79.80%	89.60%	94.59%	97.37%	98.66%	99.32%
100G	43.57%	77.81%	87.88%	94.25%	97.06%	98.44%	99.24%
1T	46.75%	70.99%	78.63%	79.27%	84.51%	79.14%	84.29%

These results are in accordance with our theoretical analysis, i.e., the asymptotic growth of the parallel compression time remained close to that represented by T_P in equation (3).

Furthermore, Table II shows the percentage decrease in compression time obtained by our solution. It can be seen that almost a 100% decrease in compression time was obtained by using 128 processes to compress 1GB, 10GB and 100GB datasets. Also the compression time was reduced close to 50% when we used 2 processes to compress datasets 100MB to 1TB. Based on this table, we generalize that every time we doubled the amount of processing units the compression time changed by a decreasing percentage of approximately $\frac{1}{p}$, validating the theoretical analysis of the computation cost T_c.

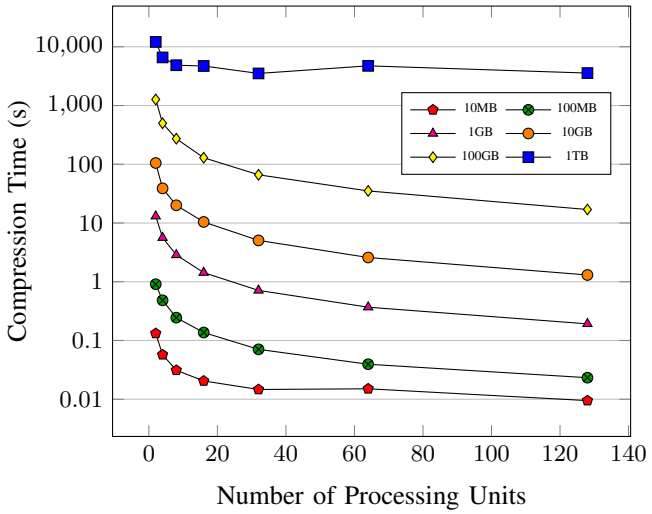

Fig. 5. Compression time (s) vs. number of processing units for paraDSRC. The Y axis is log_{10} scaled to allow large values to be displayed without distorting the visualization of the results.

Fig. 6 shows paraDSRC speedup ratio. It shows that super-linear speedups were obtained when compressing dataset

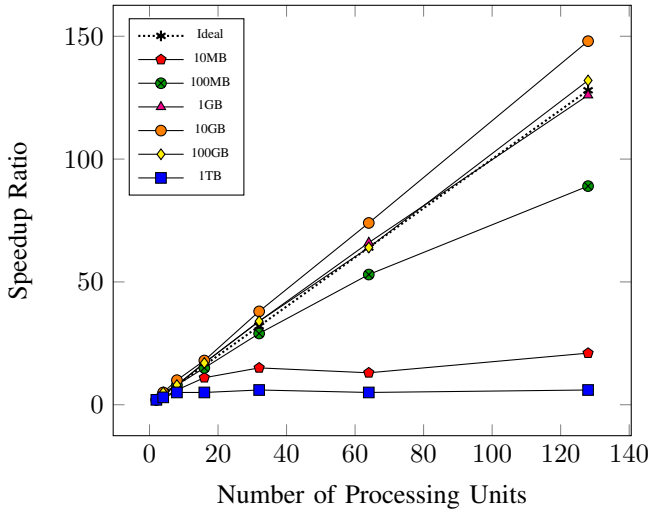

Fig. 6. Speedup ratio S for paraDSRC with increasing number of processes and different dataset sizes.

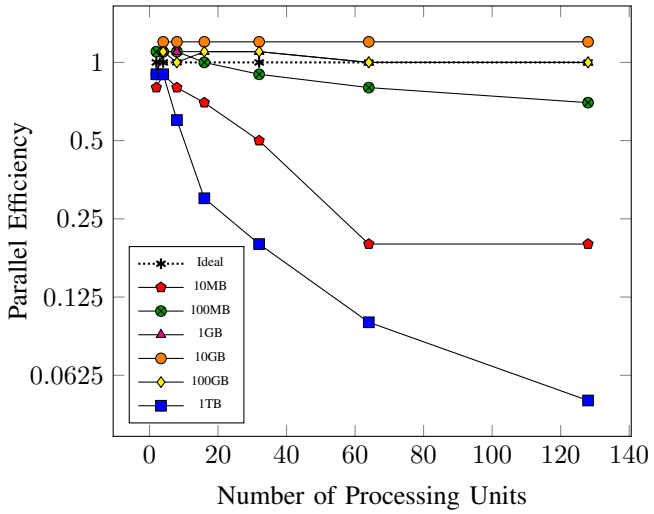

Fig. 7. Parallel efficiency E for paraDSRC with increasing number of precesses and different dataset sizes.

10GB (using 64 and 128 processes) and dataset 100GB (using 128 processes). This behavior may be due to the fact that by adding more processing units their individual cache will accumulate as well and the total cache size will increase, allowing all the amount of data being processed at a given time to fit into the new cache size, reducing the latencies incurred in memory access. This cache effect is studied in [15]. An ideal linear speedup was maintained for dataset 1GB while increasing the number of processing units. The same was the case for 100GB, remaining ideally linear by using up to 64 processes. Dataset 100MB kept a linear speedup when compressed using up to 16 processes, after the addition of more processing units the parallel time T_P increases, diminishing the speedup (which is still linear). The 10MB and 1TB datasets presented a very similar behavior, and are discussed below.

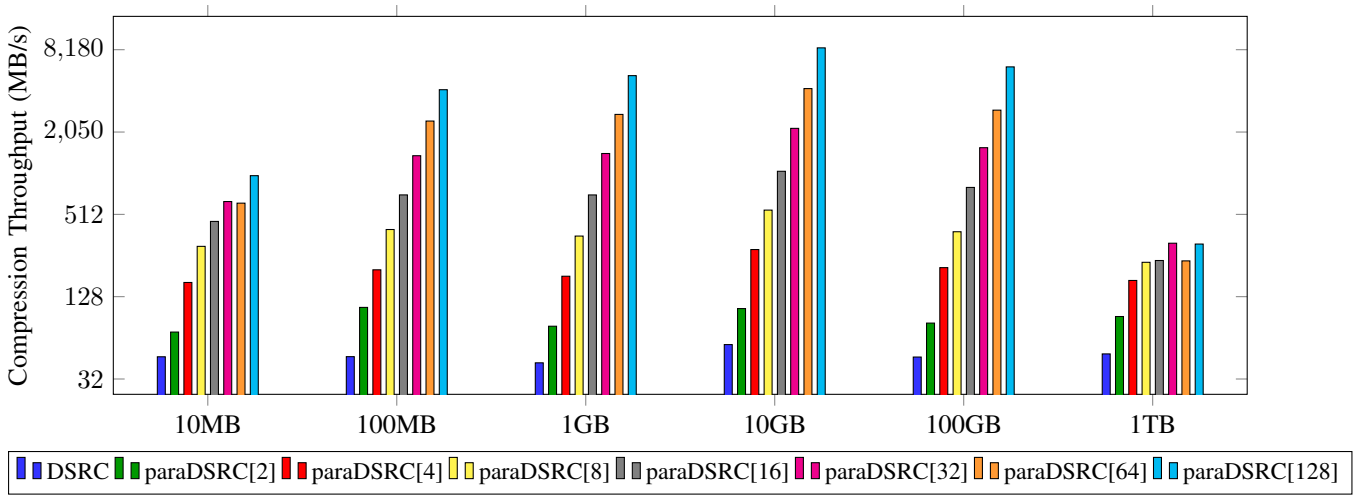

Fig. 8. Compression throughput of DSRC and paraDSRC[p], where p is the number of processes used in the test (a power of 2), for different dataset sizes.

The parallel efficiency of the algorithm was also tested and is shown in Fig. 7. Datasets 1GB, 10GB and 100GB maintained approximately a fixed efficiency while increasing the number of processing units, making the algorithm strongly scalable for these datasets [16]. Although maintaining its efficiency, dataset 100MB started to decline as more processing units were added.

Fig. 7 also shows that paraDSRC was not very efficient in computing the smallest (10MB) and the biggest (1TB) datasets of our experiments. For the 10MB dataset, as the number of processing units increased, the speedup stayed almost constant and the efficiency went down due to the fact that the communication cost T_o became much larger as compared to the computation time T_c. This behavior is typical for parallel algorithms computing very small datasets [17]. In the case of the 1TB dataset, we conclude that our moderate size cluster, although equipped with a high performance parallel file system, is not able to keep a linear speedup and hence a desired efficiency when 8 or more processing units are concurrently accessing non-contiguous regions of a big dataset[8].

We also evaluated the compression throughput of paraDSRC to have a measure of the overall performance. Within the context of this paper, compression throughput is defined as the amount of data being processed per second, expressed as megabytes per second (MB/s), that is:

$$CompressionThroughput = \frac{DatasetSize}{CompressionTime}$$

Fig. 8 shows the compression throughput results. Overall, paraDSRC presented a better compression throughput when compared to that of DSRC. Among all the datasets 10GB has the maximum throughput of 8,441MB/s (8.4GB/s) when compressed with 128 processes, which correlates with its super-linear speedup. 128 processing units were used to achieve the best compression throughput for datasets 1GB and 100GB of 5,288MB/s (5.2GB/s) and 6,128MB/s (6.1GB/s) respectively.

[8]This conclusion is based on results obtained comparing the reading times with those of reading contiguous regions of the 1TB datset

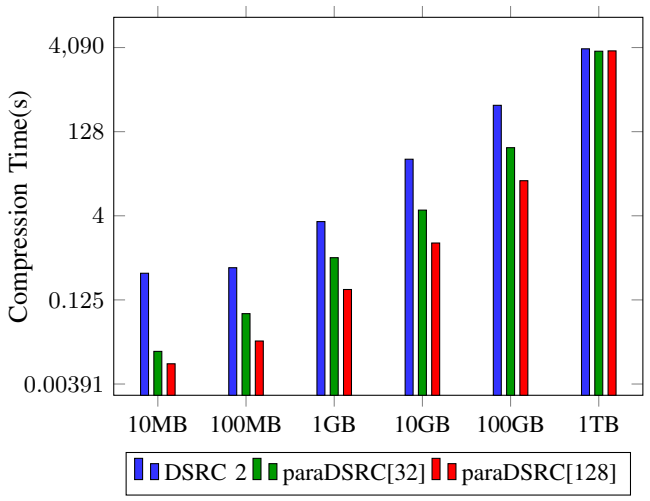

Fig. 9. Comparison between of DSRC 2 (using 32 threads) and paraDSRC (using 32 and 128 processing units), for different dataset sizes.

B. Comparison With Existing Parallel Compression Algorithms

We mentioned in the Introduction section the existence of several parallel compression algorithms such as pigz, PBZIP2, MPIBZIP2 and DSRC 2. We selected DSRC 2 for a more fair comparison against our implementation, since both algorithms are modifications of DSRC. Fig. 9 shows the compression time for DSRC 2 using 32 threads and paraDSRC using 32 and 128 processing units. We can see how paraDSRC outperforms DSRC 2 (even when 32 processes are being used) for datasets 10MB to 100GB, reaching compression times of 39.6x for some datasets using 128 processes. Table III presents the accelerations achieved by this experiment.

TABLE III
ACCELERATION OBTAINED BY $paraDSRC$ FOR 32 AND 128 PROCESSES AGAINST DSRC 2.

Datasets	paraDSRC[32]	paraDSRC[128]
10MB	25.6x	39.6x
100MB	6.7x	20.3x
1GB	4.4x	16.4x
10GB	8.1x	31.6x
100GB	5.7x	22.3x
1TB	1.1x	1.1x

V. CONCLUSION AND DISCUSSION

In this paper we presented a parallelization strategy using distributed-memory architecture for compression of big NGS datasets. Our implementation used DSRC algorithm as the underlying compression method and Message Passing Interface (MPI) in order to accelerate the compression time. To our knowledge, this is the first attempt to investigate domain decomposition strategy implemented on a memory-distributed architecture for compression of big Next Generation Sequencing datasets. The proposed strategy allowed us to devise a highly scalable parallel algorithm, which exhibited linear-speedups for most datasets and gave a minimalistic communication footprint.

A detailed description of our design was given in the paper that included the theoretical analysis of the algorithm in terms of asymptotic growth. The results obtained by our solution indicate that a decreasing percentage change of approximately $O(\frac{1}{p})$ in the compression time was possible. Furthermore, we achieve super-linear speedups for several dataset sizes while adding processing units and accelerations of up to 39x when compared with DSRC 2 (DSRC multi-threaded parallel version). Efficiency analysis indicated that paraDSRC is strongly scalable, with efficiencies being maintained in the range of $\Theta(1)$.

The experiments presented in this paper were executed on a moderate size cluster which limits our ability to process very large datasets (of the order of terabytes). As our future direction for this work we are interested in running our implementation on large clusters with parallel file systems such as XFS, PVFS or Lustre [18]–[20]. This will allow us to take advantage of the MPI-IO ROMIO[9] implementation which uses hints based on the characteristics of these parallel file systems to perform more efficient I/O requests. We hypothesize that by applying these improvements to the algorithm we will be able to obtain better compression times, speedups and efficiency for datasets greater than 1TB.

Additionally our implementation can be combined with threads in a MPI+Thread hybrid model to achieve an even faster acceleration by giving the task of analyzing and compressing the record's title, DNA sequence and quality score to several threads while the main process works in reading the input file and writing the compressed results.

[9]http://press3.mcs.anl.gov/romio

ACKNOWLEDGMENT

This work was supported in part by grants NSF CNS-1250264, NSF CNS-1441384 and NSF CCF-1464268. The authors would like to thank the members of the Parallel Computing and Data Science (PCDS) lab at Western Michigan University for their valuable input while in the process of developing this paper.

REFERENCES

[1] M. A. Quail, I. Kozarewa, F. Smith, A. Scally, P. J. Stephens, R. Durbin, H. Swerdlow, and D. J. Turner, "A large genome center's improvements to the illumina sequencing system," *Nature methods*, vol. 5, no. 12, pp. 1005–1010, 2008.

[2] J. S. Reis-Filho *et al.*, "Next-generation sequencing," *Breast Cancer Res*, vol. 11, no. Suppl 3, p. S12, 2009.

[3] D. C. Jones, W. L. Ruzzo, X. Peng, and M. G. Katze, "Compression of next-generation sequencing reads aided by highly efficient de novo assembly," *Nucleic acids research*, p. gks754, 2012.

[4] W. Tembe, J. Lowey, and E. Suh, "G-sqz: compact encoding of genomic sequence and quality data," *Bioinformatics*, vol. 26, no. 17, pp. 2192–2194, 2010.

[5] S. Deorowicz and S. Grabowski, "Compression of dna sequence reads in fastq format," *Bioinformatics*, vol. 27, no. 6, pp. 860–862, 2011.

[6] E. Grassi, F. Di Gregorio, and I. Molineris, "Kungfq: A simple and powerful approach to compress fastq files," *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, vol. 9, no. 6, pp. 1837–1842, Nov. 2012.

[7] M. Howison, "High-throughput compression of fastq data with seqdb," *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, vol. 10, no. 1, pp. 213–218, Jan. 2013.

[8] Y. J. Jeon, S. H. Park, S. M. Ahn, and H. J. Hwang, "Solidzipper: A high speed encoding method for the next-generation sequencing data," *Evolutionary bioinformatics online*, vol. 7, p. 1, 2011.

[9] P. J. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice, "The sanger fastq file format for sequences with quality scores, and the solexa/illumina fastq variants," *Nucleic acids research*, vol. 38, no. 6, pp. 1767–1771, 2010.

[10] Ł. Roguski and S. Deorowicz, "Dsrc 2industry-oriented compression of fastq files," *Bioinformatics*, vol. 30, no. 15, pp. 2213–2215, 2014.

[11] H. Na, H. Luo, W. Ting, and B. Wang, "Multi-task parallel algorithm for dsrc," *Procedia Computer Science*, vol. 31, pp. 1133–1139, 2014.

[12] B. Barney. (2010) Introduction to parallel computing. HTML. Lawrence Livermore National Laboratory. [Online]. Available: https://computing.llnl.gov/tutorials/parallel_comp

[13] A. Y. Grama, A. Gupta, and V. Kumar, "Isoefficiency: Measuring the scalability of parallel algorithms and architectures," *IEEE concurrency*, vol. 1, no. 3, pp. 12–21, 1993.

[14] G. Staples, "Torque resource manager," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006.

[15] J. Benzi and M. Damodaran, "Parallel three dimensional direct simulation monte carlo for simulating micro flows," in *Parallel Computational Fluid Dynamics 2007*. Springer, 2009, pp. 91–98.

[16] I. T. Todorov, W. Smith, K. Trachenko, and M. T. Dove, "Dl_poly_3: new dimensions in molecular dynamics simulations via massive parallelism," *Journal of Materials Chemistry*, vol. 16, no. 20, pp. 1911–1918, 2006.

[17] F. Saeed and A. Khokhar, "A domain decomposition strategy for alignment of multiple biological sequences on multiprocessor platforms," *Journal of Parallel and Distributed Computing*, vol. 69, no. 7, pp. 666–677, 2009.

[18] A. Ching, A. Choudhary, K. Coloma, W.-k. Liao, R. Ross, and W. Gropp, "Noncontiguous i/o accesses through mpi-io," in *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*. IEEE, 2003, pp. 104–111.

[19] R. Thakur, W. Gropp, and E. Lusk, "Optimizing noncontiguous accesses in mpi–io," *Parallel Computing*, vol. 28, no. 1, pp. 83–105, 2002.

[20] P. M. Dickens and J. Logan, "Y-lib: a user level library to increase the performance of mpi-io in a lustre file system environment," in *Proceedings of the 18th ACM international symposium on High performance distributed computing*. ACM, 2009, pp. 31–38.