# A Fast and Scalable Genomic Data Compressor for Multicore Clusters

Victoria Sanz[1,2] ✉, Adrián Pousa[1], Marcelo Naiouf[1], and Armando De Giusti[1,3]

[1] III-LIDI, School of Computer Sciences, National University of La Plata, Argentina
[2] CIC, Buenos Aires, Argentina
[3] CONICET, Argentina
{vsanz,apousa,mnaiouf,degiusti}@lidi.info.unlp.edu.ar

**Abstract.** Genomic data is growing rapidly due to the high demand for precision medicine. Consequently, efficient genome compression algorithms are needed to reduce storage usage in an acceptable response time. This paper introduces HybridHRCM, a hybrid MPI/OpenMP algorithm that harnesses the power of multicore clusters to compress a collection of genomic sequences. We compared our proposal with MtHRCM-opt, its multi-threaded OpenMP counterpart that is suitable for single-node multicore systems. Experimental results demonstrate that HybridHRCM enhances the scalability of MtHRCM-opt for large test collections when using the same number of cores but in a distributed way, while behaves similarly for small test collections. Furthermore, the results reveal that HybridHRCM still achieving good performance when adding more nodes, for all collections.

**Keywords:** Genomic Data Compression, Multicore Clusters, Hybrid MPI/OpenMP Parallel Programming, Performance, Scalability

## 1 Introduction

Genomic data is growing rapidly due to the high demand for precision medicine. Consequently, efficient lossless compressors for genomic sequences are needed to reduce storage usage in an acceptable response time [1, 2].

In particular, such lossless compression algorithms are classified into two categories, depending on whether they use references during compression or not: (i) reference-free algorithms compress the target sequence using only its internal characteristics; (ii) reference-based algorithms use one or more reference sequences to compress the target sequence, leveraging the high similarity between sequences of the same species [3–5]. Moreover, some reference-based methods allow compressing collections of sequences. Compared to compressing each sequence of the collection individually, batch compression carries out certain steps of the process only once and obtains a higher compression ratio [6, 7].

Related to the previously mentioned, HRCM (Hybrid Referential Compression Method) [7] is a compression algorithm for collections of genomes in FASTA format, reference-based and lossless. For each to-be-compressed sequence, the algorithm performs a first matching to find all the segments that are included in the

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_22

reference sequence. As a result, a compressed sequence is obtained, which contains information about matches and mismatches. Then, first-matching results go through a second matching, where some previously compressed sequences are taken as references (this introduces a data dependency). These second-matching results are appended to a file. After all sequences were processed, the file is compressed with 7-zip.

In order to reduce compression time, the same authors proposed MtHRCM [8], a multi-threaded implementation of HRCM. First, to satisfy the data dependency, the algorithm sequentially solves all the sequences that will be references during the second matching. Then, it uses threads to compress the remaining sequences in parallel. Each compressed output is saved in a separate intermediate file. Finally, the intermediate files are written sequentially (in order) into the final output file, which is then compressed with 7-zip. Although MtHRCM improves the performance of HRCM, it achieves a poor speedup and does not scale well due to its sequential parts and the contention at the I/O system.

Then, we proposed MtHRCM-opt [9], an optimized version of MtHRCM that reduces its sequential component. The experimental results showed that MtHRCM-opt improves the performance of MtHRCM. Also, they revealed that MtHRCM-opt scales well when increasing the number of threads/cores for smaller test collections, but the high amount of simultaneous I/O requests to disk still limits the scalability for larger test collections.

In summary, the aforementioned works focus on compressing genomic sequence collections on a single multicore machine. Single node parallelism is limited by the number of cores available and the concurrent access to shared resources (e.g. memory and disk) that may cause long latencies. In contrast, distributed parallelism allows improving the performance of some applications by leveraging the resources of multiple computers (nodes) in a cluster.

In this paper, we introduce HybridHRCM, a hybrid MPI/OpenMP algorithm based on HRCM that harnesses the power of multicore clusters to compress a collection of genomic sequences. Experimental results demonstrate that HybridHRCM enhances the scalability of MtHRCM-opt for large test collections when using the same number of cores but in a distributed way, while behaves similarly for small test collections (as expected). Furthermore, the results reveal that HybridHRCM still achieving good performance when adding more nodes (resources), for all collections.

The rest of the paper is organized as follows. Section 2 summarizes the HRCM and MtHRCM-opt algorithms. Section 3 describes our proposal. Section 4 shows our experimental results. Finally, Section 5 presents the main conclusions and future research.

## 2 Background

This section describes the HRCM and MtHRCM-opt compression algorithms. Both algorithms compress a collection of FASTA sequences with a lossless reference-based approach.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_22

### 2.1 HRCM Algorithm

HRCM [7] consists of three stages: *startup*, *matching* and *encoding*.

The *startup* extracts the reference sequence, that is, it keeps all the nucleotides (A, C, G, T) after converting them to uppercase. Then, it constructs a hash table based on the extracted reference sequence, which stores for each possible k-mer (substring of k nucleotides) all its locations in the reference sequence or -1 if it does not exist. This table is used in the matching stage.

Next, the *matching* applies these steps to each to-be-compressed sequence:

- *Extraction:* similar to the extraction step of the startup stage.
- *First-level matching:* this step iterates over the extracted sequence using a sliding window of length k. When the k-mer currently in the window appears in the reference sequence, the position and length of the longest match are recorded in the results. Otherwise, the first base of the k-mer is recorded as a mismatched character. Then, the window is slided. This iteration continues until reaching the end of the sequence.
- *Hash table construction:* only if the sequence will be a reference during the second matching, the results of the previous step along with a hash table built from their entities (taken in pairs) are stored in memory. These data structures are used in the next step. The maximum number of references for the second-level compression is configurable (parameter $L$).
- *Second-level matching:* the results of the first-level matching are compressed, using a sliding window of length 2 and taking as references the first $m$ *already compressed sequences*, where $m = min(i-1, L)$ and $i$ is the index of the current sequence. From the entities in the window, the longest match among all references is found and the information about the matched segment (sequence id, position, length) is appended to the output file. If a mismatch occurs, the first entity in the window is appended to the output file.

Finally, the *encoding* compresses the output file with 7-zip.

### 2.2 MtHRCM-opt Algorithm

MtHRCM-opt [9] uses multiple threads to compress the collection of sequences.

First, the main thread executes the *startup* stage. Then, it creates a pool of threads to perform the *matching* stage as follows.

Each thread dynamically picks the next sequence from the collection to be processed. The thread completes the first-level matching, which has no data dependence. Then, it waits for the required data structures before executing the second-level matching (i.e., both the first-level matching results and the associated hash table of the corresponding reference sequences). Specifically, let $i$ be the index of the current sequence, the thread must wait for the data structures of sequences with index between 1 and $min(i-1, L)$ to be ready. Once this condition is met, the thread completes the second-level matching and stores the results in a separate intermediate file.

After all sequences were processed, the main thread *encodes* all the intermediate files with 7-zip, resulting in a single compressed file.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_22

## 3 HybridHRCM Algorithm

HybridHRCM harnesses the power of multicore clusters to compress a collection of sequences. In particular, it relies on hybrid MPI/OpenMP programming to create a single process per machine/node with multiple threads.

All nodes *collaboratively compress* the collection of genomic sequences in stages. Each node is responsible for processing a subset of sequences that will be second-level references and a subset of the remaining sequences[1]. Let $P$ be the number of nodes, $n$ be the number of sequences in the collection and $L$ be the maximum number of references for the second-level compression, the size of the first subset is $\frac{L}{P}$ and the size of the second subset is $\frac{n-L}{P}$.

First, the main thread on each node performs the *startup* stage, which extracts the reference sequence and constructs the associated hash table. The time of this operation is negligible and its parallelization has no practical sense, for this reason the computation is replicated on each node.

In the first compression stage, intra-node threads perform the first-level matching of the sequences of the first subset (i.e. those that are second-level references). This problem is data parallel, since it only uses the information obtained in the startup. Each thread dynamically picks the next sequence from the subset, extracts it, and then computes and stores the results of the first-level matching and the associated hash table. Then, the thread communicates both data structures to the other nodes through message passing. This information is necessary for the following stages. Message reception is handled by a dedicated receiver thread on each node, which runs concurrently with the worker threads.

The second compression stage will begin once the node has received all the data structures sent by the other nodes. Having such information allows independent processing. At this stage, intra-node threads complete the processing of the sequences of the first subset. Each thread dynamically picks the next sequence from the subset, computes the second-level matching and stores the results in an independent intermediate file.

In the third compression stage, intra-node threads compress the sequences of the second subset. Each thread dynamically picks sequences, extracts each one and applies the first and second-level matching, and stores the results in an independent intermediate file.

Finally, the master process executes the *encoding* stage, which compresses all the intermediate files with 7-zip, resulting in a single file. To do this, it must access all the generated intermediate files. Consequently, each node is responsible for leaving the generated intermediate files in a location accessible by the master process (either on its local disk or on a storage).

Notice that communication time does not impact performance since the communicated data are hash tables of limited size (∼8MB each) and compressed results with high compression ratio.

[1] The algorithm assumes that each node can access the files (sequences) of both assigned subsets as well as the reference sequence file. That is, these files can be stored locally on disk or can be obtained from a storage.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_22

## 4 Experimental Results

Our experimental platform is a cluster of multicore nodes. Each node is composed of two Intel Xeon E5-2695 v4 processors (36 cores in total), 128GB RAM and a SAS disk. Hyper-Threading and Turbo Boost were disabled. All nodes are connected via 1Gb Ethernet.

Tests considered 1100 commonly used human genomes [9]: 1092 are extracted from the 1000 Genome Project; 5 are the UCSC HG16, HG17, HG18, HG19 and HG38 genomes; 2 are the Korean genomes KOREF_20090131 and KOREF_20090224; and the last is the HuRef genome. We used the UCSC HG13 genome as reference. Human genomes contain 24 chromosomes (identified as 1, 2, .., 22, X, Y) and have a size of ∼3000 MB each.

Specifically, we formed 24 test collections in total, one for each chromosome. Each test collection includes the 1100 same-numbered chromosomes from different individuals, and will be compressed against the same-numbered chromosome of HG13. This grouping allows the compressor to leverage the similarity between the to-be-compressed sequences and the reference. It is worth mentioning that: in general, the smaller the chromosome ID, the larger the collection size; each test collection includes sequences of similar size (near to the average); larger test collections are composed of larger sequences, and smaller test collections have smaller sequences. Hence, the aforementioned grouping allows us to evaluate the scalability of the algorithm.

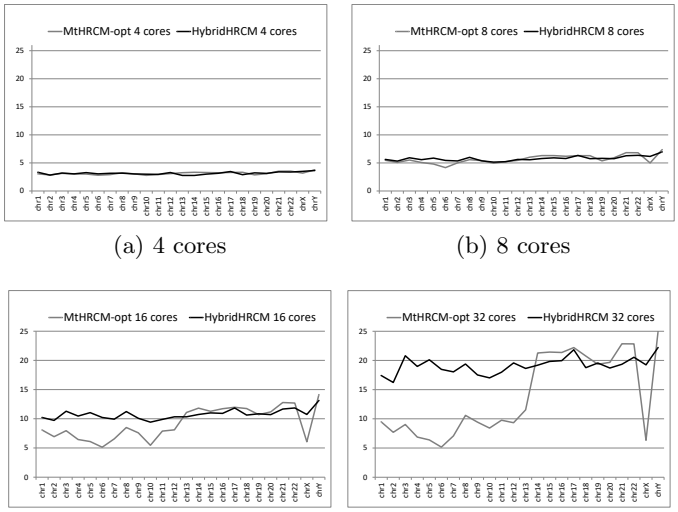
The maximum number of references for the second-level matching ($L$) was set to 275, which corresponds to 25 percent of the to-be-compressed collection.

To prove the effectiveness of our proposal, we ran HRCM (sequential code), MtHRCM-opt (multi-threaded code) and HybridHRCM (hybrid MPI/ OpenMP code) on different system configurations. In the former two cases, all the to-be-compressed sequences are stored on the local disk of the single node used. In the case of the hybrid algorithm, each cluster node stores on its local disk the to-be-compressed-sequences of its assigned subsets.

From the experimental results, we first confirmed that the three algorithms achieve the same compression efficiency. That is, for the same to-be compressed collection and the same value of L, all the algorithms obtain the same output (regardless of the system configuration used). This behavior is expected since their compression methodology is identical. In general, all test collections (a total of 3258684 MB or ∼3 TB) were compressed to 1318 MB.

Next, we verify the performance behavior of our algorithm, measured in terms of Speedup ($\frac{Time_{seq\_algorithm}}{Time_{par\_algorithm}}$). Figure 1 compares the Speedup of MtHRCM-opt and HybridHRCM, for all chromosomes, with 4, 8, 16 and 32 threads/cores in total. In HybridHRCM threads are distributed equally between two nodes (i.e. 2 nodes x 2 cores, 2 nodes x 4 cores, 2 nodes x 8 cores, 2 nodes x 16 cores). The results show that, for 4 and 8 threads, the Speedup obtained by both algorithms is similar for all chromosomes. However, for 16 and 32 threads, HybridHRCM achieves better Speedup than MtHRCM-opt for chromosomes {1..12, X} (first group), and similar Speedup for the rest of the chromosomes {13..22,Y} (second group).

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_22




(a) 4 cores (b) 8 cores

(c) 16 cores (d) 32 cores

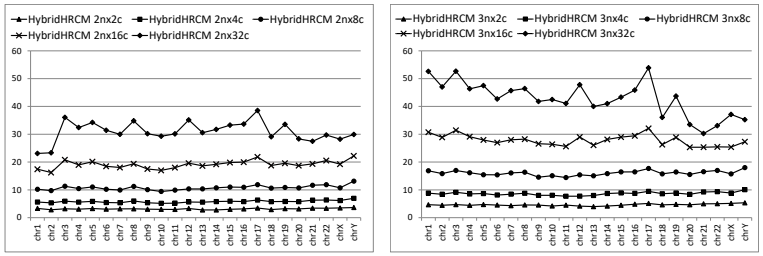
Fig. 1: Speedup comparison between MtHRCM-opt (1 node) and HybridHRCM (2 nodes), for different number of threads/cores.

To explain this behavior, we refer to our previous work [9] where we observed that the main source of overhead in MtHRCM-opt is disk contention, which limits the scalability of large collections (first group). Disk I/O is performed to extract the information of each sequence from file and write the compressed sequences to disk. For a fixed test collection (chromosome), as more threads are involved in compression, more sequences are processed in parallel, therefore there will be more I/O requests that the disk must serve simultaneously. This causes long latencies that affect performance. This overhead is even higher for large collections (first group) since they require more I/O operations. HybridHRCM distributes the to-be-compressed sequences of the collection among nodes. For this reason, the amount of work per node is lower, which implies a lower number of I/O requests to be served simultaneously (regardless of the number of threads used). This results in a performance gain, which is evident for large collections (first group) and a large number of threads.

It should be noted that, for the remaining cases, MtHRCM-opt shows good performance and is not significantly affected by disk contention. Therefore, HybridHRCM obtains similar performance when distributing the work among nodes, using the same number of total cores.

Then, we investigate the scalability of HybridHRCM when increasing the number of threads/cores per node. Figure 2 shows the Speedup of HybridHRCM, for all chromosomes, with 2 and 3 nodes, and 2, 4, 8, 16 and 32 threads/cores per node. The results in both cases reveal that, for all chromosomes, the Speedup improves as the number of cores per node increases. Also, they display that the best performance is obtained when using 32 cores per node.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_22




(a) 2 Nodes (b) 3 Nodes

Fig. 2: Speedup of HybridHRCM with 2 and 3 nodes (n) when increasing the number of threads/cores per node (c).

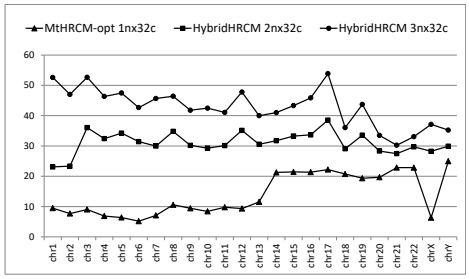



Fig. 3: Speedup of MtHRCM-opt (1 node), HybridHRCM (2 nodes) and HybridHRCM (3 nodes), with 32 threads/cores per node

In addition, we study the scalability of the algorithm when increasing the number of nodes. Figure 3 compares the Speedup of MtHRCM-opt (1 node), HybridHRCM (2 nodes) and HybridHRCM (3 nodes), with 32 threads/cores per node, since this thread configuration gave the best performance, as previously presented. Note that adding a node involves adding 32 processing cores (i.e., the total number of cores with 1, 2 and 3 nodes is 32, 64 and 96 respectively). For all chromosomes, the best performance is achieved with 3 nodes. For smaller test collections (higher ID), the performance does not increase significantly when scaling from 2 to 3 nodes. This is because smaller test collections are smaller in size and thus their compression time is lower, so as more cores are used, the overhead due to parallelism affects more severely the total time (Amdahl's law).

Finally, we analyze the Overall Throughput ($\frac{uncompressed\_data\_size\ (MB)}{compression\_time\ (seconds)}$) of the algorithms to provide a more concrete interpretation of the presented results. Throughput values were calculated considering the total size and the total compression time of all test collections (24 chromosomes), for each algorithm. HRCM (sequential code) achieves a throughput of 16.19 MB/s, MtHRCM-opt (1 node, 32 threads) obtains 144.31 MB/s, while HybridHRCM reaches 480.59 MB/s and 740.63 MB/s (with 2 and 3 nodes, 32 threads per node). As can be derived from these results, on our platform HRCM compresses all data (∼3258684 MB) in ∼56

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_22

hours, MtHRCM-opt takes ∼6h 15m, while HybridHRCM completes the compression in ∼1h 53m with 2 nodes and ∼1h 13m with 3 nodes. In other terms, per human genome (∼3000 MB) HRCM uses about 183s, MtHRCM-opt uses ∼21s, while HybridHRCM uses ∼6s and ∼4s, with 2 and 3 nodes respectively.

## 5 Conclusions and Future Work

This paper introduced HybridHRCM, a hybrid MPI/OpenMP genomic data compressor that harnesses the power of multicore clusters.

We compared HybridHRCM with MtHRCM-opt, its OpenMP counterpart for single-node systems, when using the same number of cores but in a distributed way. Experimental results show that the performance of HybridHRCM is more stable than that of MtHRCM-opt when varying the to-be compressed collection. For large test collections and a large number of cores, HybridHRCM outperforms MtHRCM-opt. For the rest of the cases, both algorithms behave similarly (as expected). Hence, HybridHRCM enhances the scalability of MtHRCM-opt.

We further studied the scalability of HybridHRCM when increasing the number of nodes (with 32 threads per node). As a main concrete result, MtHRCM-opt (1 node) compressed the whole 1100 human genomes in ∼6h 15m, HybridHRCM (2 nodes) in ∼1h 53m and HybridHRCM (3 nodes) in ∼1h 13m. Consequently, distributed computing and multicore clusters enable faster compression times for large genomic data.

In future, we plan to compare HybridHRCM with HadoopHRCM (a version of HRCM based on Hadoop MapReduce) and their advantages/disadvantages.

## References


1. National Human Genome Research Institute: Genomic Data Science. Available at: https://www.genome.gov/about-genomics/fact-sheets/Genomic-Data-Science
2. Stephens ZD. et al.: Big Data: Astronomical or Genomical?. PLoS Biology. 13(7): e1002195 (2015)
3. Kredens KV. et al.: Vertical lossless genomic data compression tools for assembled genomes: A systematic literature review. PLOS ONE 15(5): e0232942. (2020)
4. Hosseini M. et al.: A Survey on Data Compression Methods for Biological Sequences. Information. 2016; 7(4):56. (2016)
5. Wandelt S. et al.: Trends in genome compression. Curr Bioinf. 2013; 9(3):315–326. (2013)
6. Deorowicz S. et al.: GDC 2: Compression of large collections of genomes. Scientific Reports. 5, 11565 (2015).
7. Yao H,. et al.: HRCM: An Efficient Hybrid Referential Compression Method for Genomic Big Data. BioMed Research International. Vol. 2019: Article ID 3108950 (2019)
8. Yao H., et al.: Parallel compression for large collections of genomes. Concurrency Computat Pract Exper. 2022; 34(2):e6339. (2021)
9. Sanz V., et al.: Fast Genomic Data Compression on Multicore Machines. In: Naiouf M., et al. (eds). Cloud Computing, Big Data and Emerging Topics. JCC-BD&ET 2024. Communications in Computer and Information Science, vol 2189. Springer, Cham (2025).



ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_22