ParRADMeth: Identification of Differentially Methylated Regions on Multicore Clusters

Alejandro Fernández-Fraga*, Jorge González-Domínguez and Juan Touriño

Abstract—The discovery of Differentially Methylated (DM) regions is an important research field in biology, as it can help to anticipate the risk of suffering from specific diseases. Nevertheless, the high computational cost of the bioinformatic tools developed for this purpose prevents their application to large-scale datasets. Hence, much faster tools are required to further progress in this research field. In this work we present ParRADMeth, a parallel tool that applies beta-binomial regression for the identification of these DM regions. It is based on the state-of-the-art sequential tool RADMeth, which proved superior biological accuracy compared to counterparts in previous experimental evaluations. ParRADMeth provides the same DM regions as RADMeth but at significantly reduced runtime thanks to exploiting the compute capabilities of common multicore CPU clusters. For example, our tool is up to 189 times faster for real data experiments on a cluster with 16 nodes, each one containing two eight-core processors. The source code of ParRADMeth, as well as a reference manual, are available at https://github.com/UDC-GAC/ParRADMeth.

Index Terms—Differential Methylation, Bioinformatics, High Performance Computing, MPI, OpenMP

1 Introduction

Methylation is an epigenetic procedure that modifies the DNA by adding a methyl group (an alkyl derived from methane) to a DNA nucleotide. Methylation analysis is key for biologists as it is associated to different biological functions, and abnormal methylation levels can indicate the presence of certain diseases. Traditional analyses consisted in detecting Differentially Methylated (DM) sites (i.e., individual nucleotides), but the identification of DM regions (parts of the DNA dominated by DM sites) has gained attention in recent years as it provides more interesting biological insights. For instance, several studies have found DM regions related to diseases such as memory cell fate [1] or liver tumors [2].

RADMeth [3] is a cutting-edge tool to identify DM regions based on beta-binomial regression. It has demonstrated high sensitivity, specificity and control of type I errors compared to alternative tools in recent independent experimental evaluations with both synthetic and real data [4]–[6], which makes it an excellent choice for biological studies, such as [7]–[9].

• *A. Fernández-Fraga, J. González-Domínguez and J. Touriño are with the Universidade da Coruña, CITIC, Computer Architecture Group, Spain. E-mail: {a.fernandez3,jgonzalezd,juan}@udc.es*

Although analyses that identify DM regions can obtain more interesting biological insights than those only based on DM sites, this comes with a significantly higher computational cost. The main drawback of *RADMeth* and other alternative tools (see Section 2) is that they require a high runtime for large input datasets. In this paper we present *ParRADMeth*, a tool that can accelerate the identification of DM regions on modern multicore clusters. It obtains the same highly accurate biological results as *RADMeth*, but in significantly shorter time. It uses an efficient hybrid approach that combines Message Passing Interface (MPI) processes and OpenMP threads. Each MPI process launches multiple threads to efficiently exploit the cores available on each node and take advantage of the Hyperthreading technology supported by many CPU architectures.

The rest of the paper is organized as follows. Section 2 presents the state of the art related to the automatic identification of DM regions. Section 3 presents, as background, some concepts about the original *RADMeth* tool that are necessary to understand the goal of this work and the implementation of our method. Section 4 describes the parallel implementation of *ParRADMeth*. Section 5 provides the experimental evaluation in terms of runtime and scalability. Finally, concluding remarks are presented in Section 6.

2 Related Work

There has been an extensive effort for many years in the development of tools to identify DM regions [10], [11]. They can be divided into the following classes:

- Tools based on logistic regression such as the popular *methylKit* [12] or *eDMR* [13].
- Tools that assume that methylation levels of the CpG sites vary smoothly across the genome. The first tool that applied smoothing to find DM regions was *BSmooth* [14], but there exist other options such as *BiSeq* [15].
- Tools based on beta-binomial distributions, such as *MACAU* [16], *DSS-general* [17] or *GetisDMR* [18], among others. Although all these approaches use a common idea, they differ from each other in the way they estimate the regression parameters. *RADMeth* [3] can also be included in this class.
- Tools based on Hidden Markov Models (HMM), whose main advantage is that they can identify DM regions with variable size. Some examples of

this type are *HMM-Fisher* [19], *HMM-DM* [20], and *DMCHMM* [21].
- Tools based on Shannon entropy, which is a quantitative metric of change in an event series. Some examples are *QMDR* [22] and *CpG_MPs* [23].
- Tools that rely on statistical tests such as FET, ANOVA, t-test or Kruskal-Walls. Examples as *CO-HCAP* [24], *DMAP* [25] or *swDMR* [26] are flexible enough to allow the user to choose the most suitable test for each experiment, while *DMRFusion* [27] integrates several tests and decides among the results with a voting function.

Not included in these groups we can find *metilene* [28], with a circular binary segmentation algorithm to recursively divide the genome and identify DM regions.

Up to our knowledge, there is no previous work focused on accelerating the identification of DM regions with High Performance Computing (HPC) techniques. Nevertheless, we can find in the literature other bioinformatics tools that are able to exploit the computational capabilities of multicore clusters. These previous tools have been used by biologists to complete experiments on large real-world datasets in reasonable time, proving that a tool such as *ParRADMeth* can be attractive for the scientific community. Some examples are *MPIGeneNet* [29], that includes MPI routines and OpenMP directives (similarly to *ParRADMeth*) to construct genetic networks; *parMATT* [30], based on a similar approach to accelerate the alignment of multiple protein 3D-structures; *SparkBWA* [31], which uses Spark to distribute DNA sequences among workers and align them to a certain genome; or *HipMer* [32], based on the Unified Parallel C (UPC) language for genome assembling.

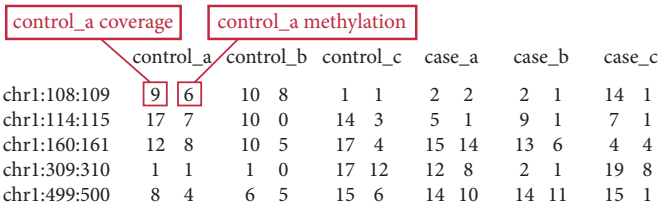
3 Background: RADMeth

RADMeth [3] is a publicly available software (as part of the *MethPipe* pipeline[1]) for computing individual DM sites and regions. The tool uses beta-binomial regression [33], a form of regression based on a beta-binomial distribution (the binomial distribution in which the probability of success at each of t trials is not fixed but randomly drawn from a beta distribution). This model allows high-precision DM analysis, it can handle medium-size experiments where it is critical to accurately model variation in methylation levels among replicates, and it accounts for influence of various experimental factors such as cell types or batch effects.

RADMeth is a command-line tool written in C++ that can receive several configuration parameters and obtains the biological input data from a text file. Concretely, this input must be a proportion table, i.e., a 2D matrix that indicates the coverage and methylation level for each CpG site (fragment of DNA where a cytosine (C) nucleotide is followed by a guanine (G) and are likely to be methylated). Figure 1 shows an example of a proportion table, where each row is dedicated to one CpG site and the columns represent the type of individuals.

RADMeth generates a single output file for each experiment. As in the input proportion table, the output file contains the information for all the CpG sites analyzed, one

1. http://smithlabresearch.org/software/methpipe/



	control_a		control_b		control_c		case_a		case_b		case_c	
chr1:108:109	9	6	10	8	1	1	2	2	2	1	14	1
chr1:114:115	17	7	10	0	14	3	5	1	9	1	7	1
chr1:160:161	12	8	10	5	17	4	15	14	13	6	4	4
chr1:309:310	1	1	1	0	17	12	12	8	2	1	19	8
chr1:499:500	8	4	6	5	15	6	14	10	14	11	15	1

Fig. 1. Example of a proportion table

line per site. Concretely, for each CpG site its line contains four columns with general information about the site (e.g., its position), one column with the p-value of the experiment for that CpG site, and four final columns that correspond to the total coverage counts and methylated read counts of the case and control groups, respectively.

The behavior of the tool can be logically divided into three phases:

1) Input phase, where data is read from input files into the appropriate structures.
2) Computation phase, where data is processed by the algorithm to produce final results.
3) Output phase, where results are written to the output text file, using one line per CpG site.

In the computation phase the regression model is fitted separately for every CpG site. In fact, to determine if one or more DM regions arise on a CpG site, two regression models are fitted: the full model and the reduced model without the test factor. The significance of DM is determined by comparing the full and the reduced models using the log-likelihood ratio test.

Algorithm 1 illustrates this computation phase. The tool starts reading each row from the input proportion table to the appropriate structure (Line 3). It calculates the coverage and methylation levels for all samples with and without the test factor (Lines 8-13). Then it checks if the CpG site has either low coverage or the same methylation levels through all samples with or without the test factor (Lines 14-17). If the CpG site passes the checks, *RADMeth* fits full and reduced regressions and analyzes their results to obtain the p-value (Lines 18-22). Finally, the results of that CpG site are written to the output file (Line 23). More information about this method can be found in [3].

4 Implementation

ParRADMeth is a novel tool to accelerate the identification of DM regions that provides exactly the same output results as *RADMeth* (and thus its high accuracy), but at significantly lower runtime thanks to exploiting the computational capabilities of multicore clusters. These parallel computers are distributed-memory systems with several nodes interconnected through a network, each of them with a memory module and several cores (see an example in Figure 2). Parallel computing on these types of systems usually follow the Single Program Multiple Data (SPMD) model, meaning that the workload is divided into different tasks that are split up among multiple processors and run simultaneously with different inputs, so that all nodes and cores cooperate

Algorithm 1: RADMeth's computation phase pseudocode

```
1  test_factor ← GetTestFactor()
2  foreach CpG site cpg_site in the proportion_table do
3      full_regresion ← WriteData(cpg_site)
4      coverage_factor ← 0
5      coverage_rest ← 0
6      methylation_level_factor ← 0
7      methylation_level_rest ← 0
       /* Agregate coverage and methylation levels */
8      foreach Sample s in the design_matrix do
9          if s has the test factor then
10             coverage_factor += full_regression[s].coverage
11             methylation_level_factor += full_regression[s].methylation_level
           else
12             coverage_rest += full_regression[s].coverage
13             methylation_level_rest += full_regression[s].methylation_level
           end
       end
       /* Calculate the p-value */
14     if zero coverage over all case or control samples then
15         p_value ← -1
16     else if methylation level is identical in all samples then
17         p_value ← -1
18     else
19         Fit(full_regression)
20         reduced_regression ← CopyWithoutTestFactor(full_regression, test_factor)
21         Fit(reduced_regression)
22         p_value ← LoglikeratioTest(reduced_regression, full_regression)
       end
23     WriteToOutputFile(cpg_site, p_value, coverage_factor, coverage_rest, methylation_level_factor,
        methylation_level_rest)
   end
```

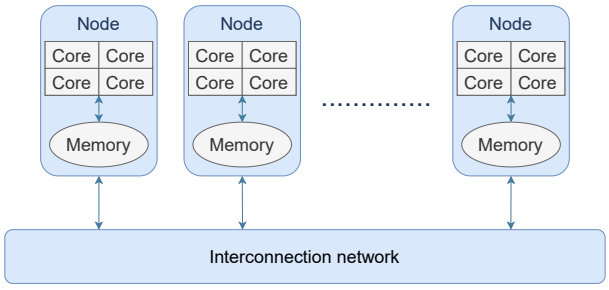


Fig. 2. Abstraction of a distributed-memory system with several cores and one memory module per node

to obtain results faster. Computational performance on a cluster depends on several factors such as the number of nodes and cores, the number of cores per node, the network features or the memory transfer rates.

A node is a unit that can be seen as a computer, that is, it is composed of main memory, processing cores, storage, and input/output system. Different nodes form the first level to distribute the workload: each node can execute different tasks of the main program. Since each node has its own memory address space, data that must be shared among nodes is sent through the network, that is why these sorts of systems are traditionally programmed with the message passing paradigm. MPI is used in *ParRADMeth* as it is the de facto standard for programming distributed-memory systems and the most widely used programming framework in the HPC community. A parallel MPI program consists of several processes, each one with associated local memory, that can communicate through the interconnection network by using send and receive routines. The MPI standard includes point-to-point and collective message passing communications, group and communicator concepts, process topologies, and parallel I/O, among other features.

Even though a pure MPI program can take advantage of all the cluster hardware by mapping one process to each core, using a hybrid approach with processes that create threads within each node has several benefits:

- Threads are lighter than processes, so creating and destroying them is faster. Also, context switching among threads of the same process is less expensive.
- Memory overhead reduction, as threads can access the same shared-memory structures, while MPI processes need a copy of the structures for each process.
- Possibility of exploiting Hyperthreading in modern CPUs, which is based on the concurrent execution of several logical threads on a single CPU core, to merge one thread instructions with the instructions of the

others, taking advantage of CPU cycles that would be free in other ways. Two concurrent threads per CPU core are common, but some processors support up to eight concurrent threads per core.

Multithreading support is included in *ParRADMeth* with OpenMP, which defines a collection of compiler directives, library routines and environment variables that implement multithreading with the fork-join model: a main thread runs the sequential parts of the program, while additional threads are forked to execute parallel tasks. Threads communicate and synchronize by using the shared memory. The main advantages of OpenMP are its portability and ease of use.

Finally, note that *ParRADMeth* uses the same configuration mechanism as *RADMeth* in order to simplify its adoption by those biologists who are already familiar with the original tool. More information about this configuration procedure can be found in the reference manual, available in the public repository of the source code of *ParRADMeth* [2].

4.1 Parallel Implementation of the Computation Phase

ParRADMeth includes two levels of parallelism in order to exploit the computational capabilities of current clusters to accelerate the identification of DM regions. First, MPI routines allow distributing data and workload among processes that can be placed on different nodes of a cluster (illustrated in Figure 2). As seen in Section 3, *RADMeth* must perform the same operations on different CpG sites (each one contained in a different line of the input file). *ParRADMeth* distributes those CpG sites among the MPI processes and, consequently, the workload is also distributed. Specifically, the proportion table is split into equal-size blocks (consecutive lines are assigned to the same process), so each process only computes its corresponding part.

The original *RADMeth* works with rows one by one, i.e., it reads one row, performs the computations related to it, writes the results to the output file, and goes to the next line. This means that only one row is kept in memory at a time. Although this is the best choice in terms of memory requirements, it may not be the best option in terms of execution time, especially for parallel computing, since it will force the tool to be continuously accessing the file. That is why *ParRADMeth* comes with a technique that we have named *AllInOneGo File Processing* to process the input proportion table.

This technique consists in reading the whole file at once and keeping the whole proportion table in memory before starting with the computations in order to reduce cache misses. Concretely, each process will only save into its local memory the rows of the input file that are assigned to it. The *AllInOneGo File Processing* is also applied to the output: instead of writing the regions obtained for each CpG site one by one, an array is created in the local memory of the MPI process, and it is used to save the resultant information of all the CpG sites assigned to the process. When the process finishes with the whole block, all the results are written at once to the output file. More information about how *ParRADMeth* efficiently deals with input/output files is provided in Subsection 4.2.

2. https://github.com/UDC-GAC/ParRADMeth

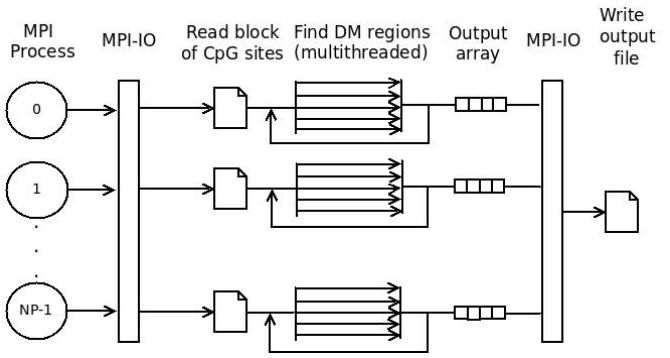


Fig. 3. Workflow of *ParRADMeth*

In addition, the values of the rows of the proportion table are stored as `size_t` (in general, 32 or 64 bits long) in *RADMeth*, which seems unnecessarily large because the range of values is limited and small, so they can be represented with just 16 bits. To alleviate the memory requirements of the *AllInOneGo File Processing*, the data type was changed to `unsigned int`, which is sufficient to represent the information of the proportion table.

However, the pure block data distribution applied to the MPI processes comes with a workload imbalance as there is a huge variation among the workload associated to different CpG sites. On the one hand, some rows are extreme cases that do not need to execute the fitting phase (see Algorithm 1, Lines 14-17). On the other hand, the fitting algorithm does not take the same time to execute on different data. This leads to situations where even if the number of lines in the input dataset is fairly distributed the workload is not. This is the reason why *ParRADMeth* does not use a pure MPI parallelization, with one process per CPU core. Instead, each process is related to a group of cores and launches several OpenMP threads (one per core). In this case the CpG sites that were previously assigned to a certain MPI process are distributed among the associated threads in a dynamic way (new sites are assigned once threads finish the processing of the previous ones). Each thread can directly access the part of the proportion table that was stored in the local memory belonging to its parent MPI process, without need for synchronization. Furthermore, the threads do not need any synchronization to store the resulting DM regions obtained for each CpG site in the output array created with the *AllInOneGo File Processing*, as each thread always works on different CpG sites (i.e., on different indexes of the output array).

Figure 3 provides a graphical overview of the two-level parallel implementation applied in *ParRADMeth*. Note that this hybrid MPI/OpenMP approach is able to exploit Hyperthreading on CPUs (where available).

4.2 Parallel Input/Output

As explained in the previous subsection, the *AllInOneGo File Processing* technique included in *ParRADMeth* makes each MPI process read a block of CpG sites (rows in the input file) and write its resulting DM regions at once. In

Algorithm 2: *ParRADMeth*'s parallel input phase pseudocode

Input: A string, $path\_to\_input\_file$, containing the path to the proportion table
Output: A string, $input\_block\_string$, containing the input lines for each process

```
1  input_file ← MPI_File_open(path_to_input_file)
   /* Figure out who reads what */
2  filesize ← MPI_File_get_size(input_file)
3  start_offset ← CalculateStartOffset(filesize)
4  end_offset ← CalculateEndOffset(filesize)
5  overlap ← CalculateLineLength(input_file)
6  end_offset += overlap
7  input_block ← MPI_File_read_at_all(input_file, start_offset, end_offset)
   /* Avoid half lines at the start and at the end and ensure no two processes keep the
      same line in their buffers */
8  true_start ← 0
9  while input_block[true_start] is not a newline character do true_start++
10 true_start++
11 true_end ← end_offset - start_offset - overlap
12 while input_block[true_end] is not a newline character do true_end++
13 input_block[true_end + 1] ← '\0'
14 input_block_string = string(input_block[true_start])
15 MPI_File_close(input_file)
```

a preliminary version of *ParRADMeth* there was a main process in charge of reading the input proportion table and distributing the data among the other processes, as well as gathering their results and writing them to the output file. It means that the phases of reading the input and writing the output were sequential. A preliminary benchmarking of this version pointed out that, after having applied an efficient data distribution with MPI processes and OpenMP threads, these phases were a bottleneck that degraded the overall performance of the program. Therefore, these I/O phases were redesigned with parallel computing in mind.

First, MPI-IO functions were used to parallelize the input phase, allowing each process to read from a certain offset. The input format (see Figure 1) is very appropriate for parallel processing, since we are interested in having a block of consecutive CpG sites (rows) in each process and that is the way they are physically stored in the file. Therefore, in *ParRADMeth* each process only reads the block of rows that it will process instead of reading the whole file.

However, this approach is not so straightforward as not all rows have the same length and the number of rows in a file is not known in advance. Algorithm 2 shows the pseudocode of the parallel input. *ParRADMeth* uses the function `MPI_File_get_size` (Line 2), to know in advance the size of the input file (in bytes) and then distribute bytes among processes to create a fair distribution of rows (Lines 3 and 4). To avoid one line to be split between two processes, making none of them able to compute it correctly, an overlapping technique was also implemented: assuming p processes, process $n \in [0, p\text{-}1]$ reads its block and extra final bytes to ensure that it will be able to correctly process the row that it may share with process n+1 (Lines 5 to 7). Some of these overlapped bytes at the start and at the end of the block belong to rows assigned to neighbor processes, so process n must adjust the real limits of its block to avoid processing the same row twice by different processes (Lines 8 to 14).

A similar version that used Unix I/O to read the input in parallel was also implemented. However, it was discarded after a preliminary benchmarking showed that it obtained significantly worse performance than using MPI-IO.

Regarding the output file, as explained in Section 3, each row in the input proportion table produces a row in the output file, so after a certain process computes all consecutive rows on its input block, it will generate an output block of also consecutive rows. The *AllInOneGo File Processing* technique explained in the previous subsection saves these results in an array. Note that the output blocks must be written in order (i.e., block of process n just after the block of process $n-1$). Algorithm 3 shows how *ParRADMeth* writes the DM regions to the output file. All processes need to know the size of the output block generated by the previous processes to calculate its offset in the file. The `MPI_Allgather` collective is used to share this information among all processes (Line 2). Then, this offset is used to write in the correct position of the output file with the `MPI_File_write_at_all` routine (Line 5).

5 Experimental Evaluation

The experimental evaluation of *ParRADMeth* has been performed in terms of execution time and scalability, as our tool provides the same DM regions as *RADMeth*. Previous works have already proved the high accuracy of the original tool compared with the state of the art [4]–[6]. These results are completely valid, thus, for *ParRADMeth*. The speedup S is used in this section as a measure of performance scalability. It is calculated as the acceleration by comparing the sequential time Ts and the parallel time using n cores Tn (either with threads and/or processes): $S(n) = \frac{Ts}{Tn}$. A parallel application is scalable when the speedup increases with the number of cores. The closer $S(n)$ is to n, the better parallel scalability the algorithm presents.

This section provides a performance comparison of *ParRADMeth* versus *RADMeth* on a 16-node cluster with 256

Algorithm 3: *ParRADMeth*'s parallel output phase pseudocode

Input: A string, my_output_block, containing the output block of the process
A string, $path\_to\_output\_file$, containing a path to the output file
Output: The output file with every output block correctly written

```
1 my_output_length ← my_output_block.length()
  /* Gather all output blocks lengths in every process */
2 output_blocks_lengths ← MPI_Allgather(my_output_length)
3 my_write_offset ← CalculateOffset(output_blocks_lengths)
  /* Write in parallel to the output file */
4 output_file ← MPI_File_open(path_to_output_file)
5 MPI_File_write_at_all(output_file, my_output_block, my_write_offset)
6 MPI_File_close(output_file)
```

TABLE 1
Datasets specification

Dataset	#CpG sites	#Samples	Size prop. table	Seq. time
Akalin	28,670,426	2	823 MB	10,886 s
Heyn	28,299,639	2	869 MB	18,736 s
Berman	28,149,963	2	880 MB	42,800 s
Hansen	28,217,449	6	1.4 GB	45,931 s

CPU cores (16 cores per node). Each node has two octa-core Intel Xeon E5-2660 Sandy Bridge-EP processors which support Hyperthreading (up to two logical threads per CPU core), and 64 GB of memory. The nodes are interconnected through a low-latency and high-bandwidth InfiniBand FDR network. Regarding software, both *RADMeth* and *ParRADMeth* were compiled with the GNU GCC compiler v.8.3.0, and the latter is linked to the OpenMPI library v.3.1.4.

Four different real biological datasets were used for this experimental evaluation. Table 1 summarizes the characteristics of these datasets, which are named according to the first author of the experiment where they were published. The *Akalin* dataset is used to compare methylomas of HCT116 cells with those of cells cloned without DNMT1 and DNMT3b [34]. The *Heyn* dataset provides information about centenarians and newborns, which can be very useful to analyze the differences in methylation between these two extremely different individuals [35]. *Berman* shows information of individuals with and without colorectal cancer [36], while the *Hansen* dataset compares cells immortalized with EBV virus with others activated with CD40 [37]. Table 1 includes two columns with the size of the input proportion table and the sequential time required by *RADMeth* to analyze these datasets, which can be more than 12 hours.

5.1 Experiments on One Node

The hybrid MPI/OpenMP implementation included in *ParRADMeth* allows the user to choose among different configurations of number of processes and threads. Before starting testing the scalability of the tool it is necessary to select the configuration that provides the best performance in one single node and this configuration will be assumed as the best one when increasing the number of nodes.

As was already mentioned in Section 3, each row of the input file may take a different time to be processed. The data distribution applied to the MPI processes does not take into account this variability, as it always provides the same amount of CpG sites (rows in the input file) to each process. The impact on performance of this workload imbalance can be alleviated with the use of several OpenMP threads per process. Figure 4 shows the speedups obtained by the hybrid MPI/OpenMP parallel implementation of *ParRADMeth* when using one MPI process, variable number of threads (up to 32 with Hyperthreading) and the `static`, `dynamic` and `guided` OpenMP scheduling policies. The baseline is the sequential execution time of *RADMeth*. It can be seen that the `dynamic` schedule achieves the best performance for all experiments. In all datasets except *Akalin* it is closely followed by the `guided` schedule. The reason is that this dataset has an isolated block of very high computational demanding rows, and when using `guided` or `static` schedules this whole block is assigned to the same thread.

The high speedups shown in Figure 4 prove that a configuration with only one MPI process with 16 or 32 threads (depending on the use of Hyperthreading) and the `dynamic` policy is adequate, and thus it will be used for the following experiments when working with the hybrid implementation.

5.2 Experiments for Scalability

Figure 5 shows the speedups over the sequential tool *RADMeth* of the three following configurations of *ParRADMeth*, for a varying number of nodes (16 cores per node):

- One MPI process per CPU core, without spawning any OpenMP thread (pure MPI version).
- One MPI process per node, each one with 16 threads associated and using a `dynamic` scheduling policy (hybrid version).
- One MPI process per node, each one spawning 32 threads with a `dynamic` scheduling policy (hybrid with Hypethreading).

The first conclusion that can be drawn is that executions with processes launching several OpenMP threads with the `dynamic` schedule get higher speedups than those with only MPI processes. On average, these hybrid executions are 4.47 and 3.69 times faster than the pure MPI one, with or without Hyperthreading, respectively. The difference is more significant for the *Akalin* dataset where the workload is mainly concentrated in a small block of rows and thus it is completely unbalanced. With this dataset the use of threads and Hyperthreading reduces the runtime by a factor

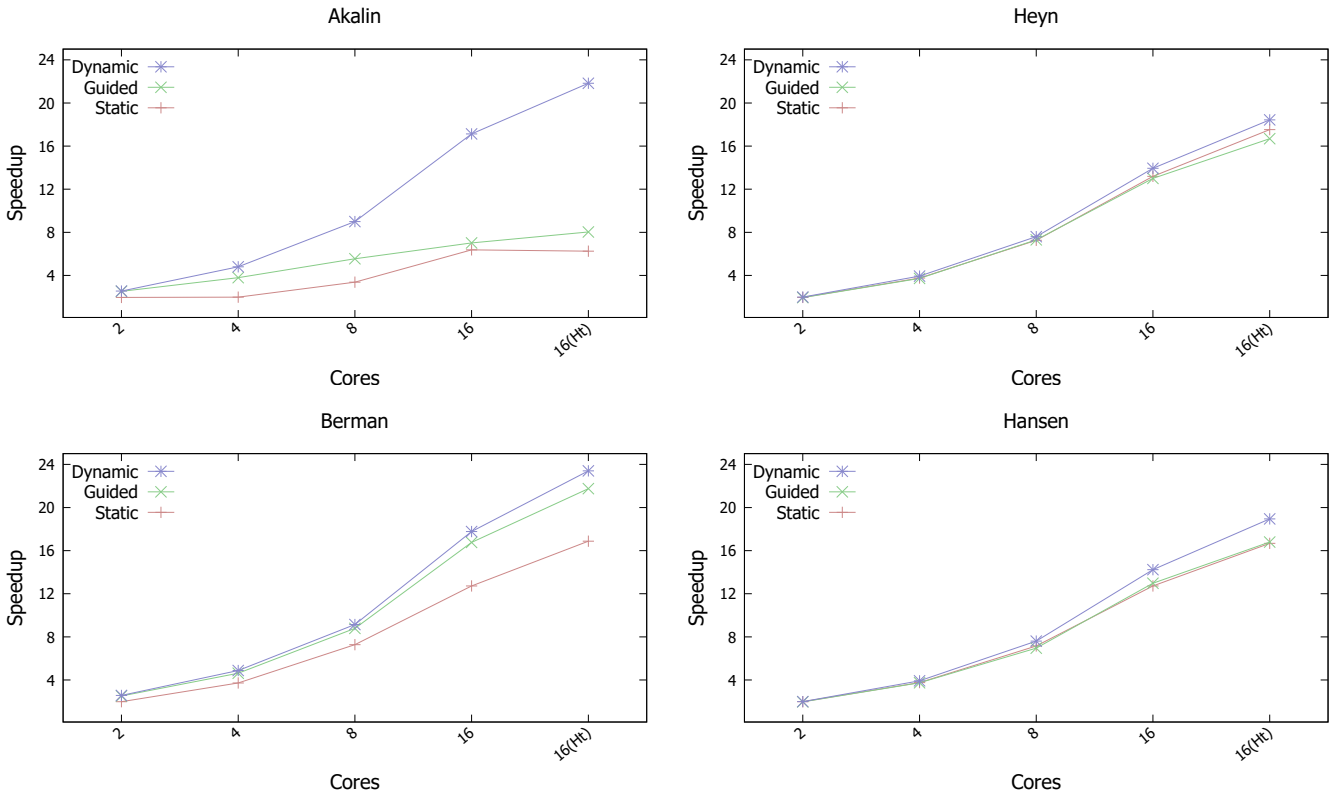


Fig. 4. Speedup of *ParRADMeth* over *RADMeth* on one node. *ParRADMeth* uses one MPI process, a varying number of threads and the three scheduling options

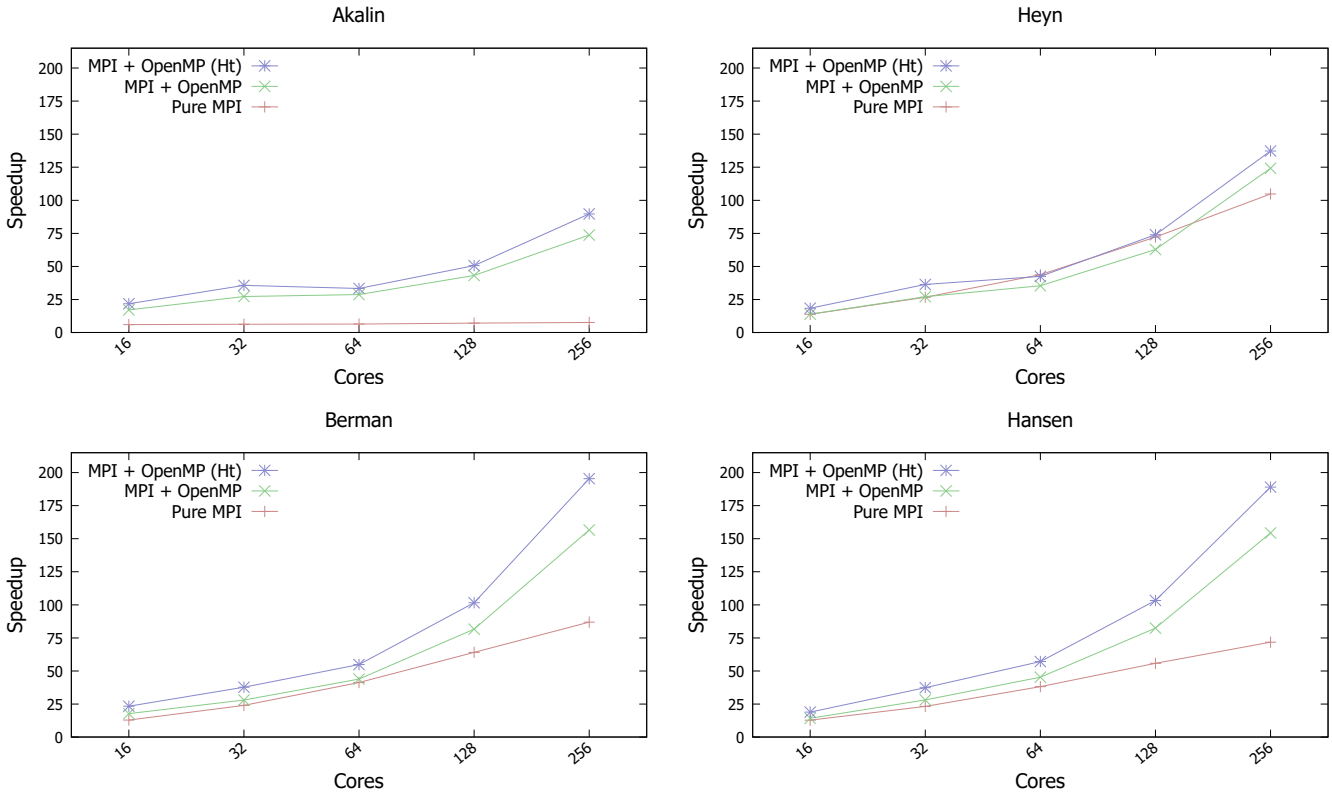


Fig. 5. Speedup of *ParRADMeth* over *RADMeth* using three versions of the parallel implementation and a varying number of nodes

TABLE 2
Execution times for different versions of the tool (in seconds)

Dataset	RADMeth	ParRADMeth		
	1 core	1 core (Ht)	256 cores	256 cores (Ht)
Akalin	10,886	8,186	147	121
Heyn	18,736	13,968	150	136
Berman	42,800	31,629	273	219
Hansen	45,931	34,224	297	242

of 11.70 compared to pure MPI execution, which is never able to achieve a speedup greater than eight, even working on up to 256 cores. This proves that a mechanism to alleviate the impact of the workload imbalance among different CpG sites (rows of the input file) is compulsory in order to achieve high performance.

These experimental results prove that *ParRADMeth* can be useful for scientists to dramatically reduce the runtime needed to find DM regions. Table 2 provides a summary of this runtime reduction. It shows that *ParRADMeth* is faster than the original *RADMeth* even using the same hardware (one core). There are two reasons. One, *ParRADMeth* is able to exploit Hyperthreading on that core by launching two logical threads. And two, the *AllInOneGo File Processing* technique presented in Subsection 4.1 is also beneficial when working only with one core, as it prevents alternating computation with disk access, thus reducing the associated overhead through a more effective use of the memory hierarchy. Furthermore, the exploitation of the 16 nodes of the cluster, including the Hyperthreading technique, allows reducing the runtime of the most expensive dataset (*Hansen*) from 12 hours and 45 minutes to only 4 minutes (189x speedup). Finally, *ParRADMeth* is also beneficial for extremely unbalanced datasets such as *Akalin*, reducing the runtime from 3 hours to 2 minutes.

6 Conclusions

Currently, one interesting goal in DNA methylation studies consists in detecting DM regions under different biological conditions, which can help to better understand the function of the methylation process. However, these analyses may take a huge time for large or even medium size datasets. In this work we have presented *ParRADMeth*, a parallel application that obtains the same biological results as the previously tested *RADMeth* tool, but at reduced runtime thanks to exploiting the hardware of multicore clusters.

ParRADMeth is based on a hybrid MPI/OpenMP parallel implementation. On the one hand, the MPI routines allow the execution on several nodes of a distributed-memory system by assigning a similar number of CpG sites per process. On the other hand, the use of several OpenMP threads per process reduces the impact of the workload imbalance among different CpG sites, which can be extremely important in real scenarios. The experimental evaluation was performed on a cluster with 16 nodes, each one with 16 CPU cores (a total of 256 cores), using four representative datasets with real biological data and different characteristics. For instance, one dataset presents an extreme case where most of the computational workload is generated by a few CpG sites. *ParRADMeth* is faster than *RADMeth* in all scenarios, even using the same hardware resources. Its impact is more remarkable for a large number of resources, being able to reduce an execution from more than a half day (12 hours and 45 minutes) to only 4 minutes.

As future work our plan is to apply similar parallel approaches to other stages of the *MethPipe* pipeline, so the different stages could be integrated in order to exploit altogether the resources of a multicore cluster.

Acknowledgment

This work was supported by the Ministry of Science and Innovation of Spain (PID2019-104184RB-I00 / AEI / 10.13039/501100011033), and by Xunta de Galicia and FEDER funds (Centro de Investigación de Galicia accreditation 2019-2022 and Consolidation Program of Competitive Reference Groups, under Grants ED431G 2019/01 and ED431C 2021/30, respectively).

References

[1] S. A. Carty, M. Gohil, L. B. Banks, R. M. Cotton, M. E. Johnson, E. Stelekati, A. D. Wells, E. J. Wherry, G. A. Koretzky, and M. S. Jordan, "The loss of TET2 promotes CD8+ T cell memory differentiation," *The Journal of Immunology*, vol. 200, no. 1, pp. 82–91, 2018.

[2] J. Matsushita, K. Okamura, K. Nakabayashi, T. Suzuki, Y. Horibe, T. Kawai, T. Sakurai, S. Yamashita, Y. Higami, G. Ichihara *et al.*, "The DNA methylation profile of liver tumors in C3H mice and identification of differentially methylated regions involved in the regulation of tumorigenic genes," *BMC Cancer*, vol. 18, no. 1, pp. 1–15, 2018.

[3] E. Dolzhenko and A. D. Smith, "Using beta-binomial regression for high-precision differential methylation analysis in multifactor whole-genome bisulfite sequencing experiments," *BMC Bioinformatics*, vol. 15, no. 1, pp. 1–8, 2014.

[4] H.-U. Klein and K. Hebestreit, "An evaluation of methods to test predefined genomic regions for differential methylation in bisulfite sequencing data," *Briefings in Bioinformatics*, vol. 17, no. 5, pp. 796–807, 2016.

[5] C. Han, H. Tang, S. Lou, Y. Gao, M. H. Cho, and S. Lin, "Evaluation of recent statistical methods for detecting differential methylation using BS-seq data," *OBM Genetics*, vol. 2, no. 4, pp. 1–1, 2018.

[6] I. Huh, X. Wu, T. Park, and S. V. Yi, "Detecting differential DNA methylation from sequencing of bisulfite converted DNA of diverse species," *Briefings in Bioinformatics*, vol. 20, no. 1, pp. 33–46, 2019.

[7] R. J. Lund, M. Kyläniemi, N. Pettersson, R. Kaukonen, M. Konki, N. M. Scheinin, L. Karlsson, H. Karlsson, and E. Ekholm, "Placental DNA methylation marks are associated with maternal depressive symptoms during early pregnancy," *Neurobiology of Stress*, vol. 15, p. 100374, 2021.

[8] X. Wu, A. R. Lindsey, P. Chatterjee, J. H. Werren, R. Stouthamer, and S. V. Yi, "Distinct epigenomic and transcriptomic modifications associated with wolbachia-mediated asexuality," *PLoS Pathogens*, vol. 16, no. 3, p. e1008397, 2020.

[9] L. T. Ong, S. D. Schibeci, N. L. Fewings, D. R. Booth, and G. P. Parnell, "Age-dependent VDR peak DNA methylation as a mechanism for latitude-dependent multiple sclerosis risk," *Epigenetics & Chromatin*, vol. 14, no. 1, pp. 1–12, 2021.

[10] M. D. Robinson, A. Kahraman, C. W. Law, H. Lindsay, M. Nowicka, L. M. Weber, and X. Zhou, "Statistical methods for detecting differentially methylated loci and regions," *Frontiers in Genetics*, vol. 5, p. 324, 2014.

[11] A. Shafi, C. Mitrea, T. Nguyen, and S. Draghici, "A survey of the approaches for identifying differential methylation using bisulfite sequencing data," *Briefings in Bioinformatics*, vol. 19, no. 5, pp. 737–753, 2018.

[12] A. Akalin, M. Kormaksson, S. Li, F. E. Garrett-Bakelman, M. E. Figueroa, A. Melnick, and C. E. Mason, "methylKit: a comprehensive R package for the analysis of genome-wide DNA methylation profiles," *Genome Biology*, vol. 13, no. 10, pp. 1–9, 2012.

[13] S. Li, F. E. Garrett-Bakelman, A. Akalin, P. Zumbo, R. Levine, B. L. To, I. D. Lewis, A. L. Brown, R. J. D'Andrea, A. Melnick *et al.*, "An optimized algorithm for detecting and annotating regional differential methylation," *BMC Bioinformatics*, vol. 14, no. 5, pp. 1–9, 2013.

[14] K. D. Hansen, B. Langmead, and R. A. Irizarry, "BSmooth: from whole genome bisulfite sequencing reads to differentially methylated regions," *Genome Biology*, vol. 13, no. 10, pp. 1–10, 2012.

[15] K. Hebestreit, M. Dugas, and H.-U. Klein, "Detection of significantly differentially methylated regions in targeted bisulfite sequencing data," *Bioinformatics*, vol. 29, no. 13, pp. 1647–1653, 2013.

[16] A. J. Lea, J. Tung, and X. Zhou, "A flexible, efficient binomial mixed model for identifying differential DNA methylation in bisulfite sequencing data," *PLoS Genetics*, vol. 11, no. 11, p. e1005650, 2015.

[17] Y. Park and H. Wu, "Differential methylation analysis for BS-seq data under general experimental design," *Bioinformatics*, vol. 32, no. 10, pp. 1446–1453, 2016.

[18] Y. Wen, F. Chen, Q. Zhang, Y. Zhuang, and Z. Li, "Detection of differentially methylated regions in whole genome bisulfite sequencing data using local Getis-Ord statistics," *Bioinformatics*, vol. 32, no. 22, pp. 3396–3404, 2016.

[19] S. Sun and X. Yu, "HMM-Fisher: identifying differential methylation using a hidden Markov model and Fisher's exact test," *Statistical Applications in Genetics and Molecular Biology*, vol. 15, no. 1, pp. 55–67, 2016.

[20] X. Yu and S. Sun, "HMM-DM: identifying differentially methylated regions using a hidden Markov model," *Statistical Applications in Genetics and Molecular Biology*, vol. 15, no. 1, pp. 69–81, 2016.

[21] F. Shokoohi, D. A. Stephens, G. Bourque, T. Pastinen, C. M. Greenwood, and A. Labbe, "A hidden Markov model for identifying differentially methylated sites in bisulfite sequencing data," *Biometrics*, vol. 75, no. 1, pp. 210–221, 2019.

[22] Y. Zhang, H. Liu, J. Lv, X. Xiao, J. Zhu, X. Liu, J. Su, X. Li, Q. Wu, F. Wang *et al.*, "QDMR: a quantitative method for identification of differentially methylated regions by entropy," *Nucleic Acids Research*, vol. 39, no. 9, p. e58, 2011.

[23] J. Su, H. Yan, Y. Wei, H. Liu, H. Liu, F. Wang, J. Lv, Q. Wu, and Y. Zhang, "CpG_MPs: identification of CpG methylation patterns of genomic regions from high-throughput bisulfite sequencing data," *Nucleic Acids Research*, vol. 41, no. 1, p. e4, 2013.

[24] C. D. Warden, H. Lee, J. D. Tompkins, X. Li, C. Wang, A. D. Riggs, H. Yu, R. Jove, and Y.-C. Yuan, "COHCAP: an integrative genomic pipeline for single-nucleotide resolution DNA methylation analysis," *Nucleic Acids Research*, vol. 41, no. 11, p. e117, 2013.

[25] P. A. Stockwell, A. Chatterjee, E. J. Rodger, and I. M. Morison, "DMAP: differential methylation analysis package for RRBS and WGBS data," *Bioinformatics*, vol. 30, no. 13, pp. 1814–1822, 2014.

[26] Z. Wang, X. Li, Y. Jiang, Q. Shao, Q. Liu, B. Chen, and D. Huang, "swDMR: a sliding window approach to identify differentially methylated regions based on whole genome bisulfite sequencing," *PloS One*, vol. 10, no. 7, p. e0132866, 2015.

[27] M. Yassi, E. S. Davodly, A. M. Shariatpanahi, M. Heidari, M. Dayyani, A. Heravi-Moussavi, M. H. Moattar, and M. A. Kerachian, "DMRFusion: a differentially methylated region detection tool based on the ranked fusion method," *Genomics*, vol. 110, no. 6, pp. 366–374, 2018.

[28] F. Jühling, H. Kretzmer, S. H. Bernhart, C. Otto, P. F. Stadler, and S. Hoffmann, "metilene: fast and sensitive calling of differentially methylated regions from bisulfite sequencing data," *Genome Research*, vol. 26, no. 2, pp. 256–262, 2016.

[29] J. Gonzalez-Dominguez and M. J. Martin, "MPIGeneNet: Parallel calculation of gene co-expression networks on multicore clusters," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 15, no. 5, pp. 1732–1737, 2017.

[30] M. V. Shegay, D. A. Suplatov, N. N. Popova, V. K. Švedas, and V. V. Voevodin, "parMATT: parallel multiple alignment of protein 3D-structures with translations and twists for distributed-memory systems," *Bioinformatics*, vol. 35, no. 21, pp. 4456–4458, 2019.

[31] J. M. Abuín, J. C. Pichel, T. F. Pena, and J. Amigo, "SparkBWA: speeding up the alignment of high-throughput DNA sequencing data," *PloS One*, vol. 11, no. 5, p. e0155461, 2016.

[32] E. Georganas, A. Buluç, J. Chapman, S. Hofmeyr, C. Aluru, R. Egan, L. Oliker, D. Rokhsar, and K. Yelick, "HipMer: an extreme-scale de novo genome assembler," in *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015, pp. 1–11.

[33] M. J. Crowder, "Beta-binomial ANOVA for proportions," *Applied Statistics*, vol. 27, no. 1, pp. 34–37, 1978.

[34] A. Akalin, F. E. Garrett-Bakelman, M. Kormaksson, J. Busuttil, L. Zhang, I. Khrebtukova, T. A. Milne, Y. Huang, D. Biswas, J. L. Hess *et al.*, "Base-pair resolution DNA methylation sequencing reveals profoundly divergent epigenetic landscapes in acute myeloid leukemia," *PLoS Genetics*, vol. 8, no. 6, p. e1002781, 2012.

[35] H. Heyn, N. Li, H. J. Ferreira, S. Moran, D. G. Pisano, A. Gomez, J. Diez, J. V. Sanchez-Mut, F. Setien, F. J. Carmona *et al.*, "Distinct DNA methylomes of newborns and centenarians," *Proceedings of the National Academy of Sciences*, vol. 109, no. 26, pp. 10 522–10 527, 2012.

[36] B. P. Berman, D. J. Weisenberger, J. F. Aman, T. Hinoue, Z. Ramjan, Y. Liu, H. Noushmehr, C. P. Lange, C. M. van Dijk, R. A. Tollenaar *et al.*, "Regions of focal DNA hypermethylation and long-range hypomethylation in colorectal cancer coincide with nuclear lamina–associated domains," *Nature Genetics*, vol. 44, no. 1, pp. 40–46, 2012.

[37] K. D. Hansen, S. Sabunciyan, B. Langmead, N. Nagy, R. Curley, G. Klein, E. Klein, D. Salamon, and A. P. Feinberg, "Large-scale hypomethylated blocks associated with Epstein-Barr virus–induced B-cell immortalization," *Genome Research*, vol. 24, no. 2, pp. 177–184, 2014.

Alejandro Fernández-Fraga received the B.S. in computer science from the Universidade da Coruña (UDC), Spain, in 2021, where he is currently pursuing a Ph.D. He also holds an M.S. in High Performance Computing from UDC since 2022. His research interests are related to the acceleration of bioinformatic tools using HPC techniques.

Jorge González-Domínguez received the B.S., M.S., and Ph.D. degrees in computer science from the Universidade da Coruña (UDC), Spain, in 2008, 2009, and 2013, respectively. He is currently an Associate Professor with the Department of Computer Engineering, UDC. His main research interests include the development of parallel applications on multiple fields, such as bioinformatics, data mining, and machine learning, focused on different architectures (multicore systems, GPUs, clusters, and so on). His homepage is http://gac.udc.es/~jorgeg

Juan Touriño is a Full Professor with the Department of Computer Engineering, Universidade da Coruña, where he leads the Computer Architecture Group. He has extensively published in the area of HPC: HPC in Bioinformatics, HPC & Big Data convergence, high performance architectures and networks, HPC programming languages and compilers, parallel algorithms and applications. He is coauthor of more than 170 papers on these topics in international conferences and journals. His homepage is http://gac.udc.es/~juan.