Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

A Hybrid Implementation of Massively Parallel Multiple Sequence Alignment Method Based on Artificial Bee Colony Algorithm

A. Charalampidou[a,b]*, P. Daoglou[a,b], D. Folias[a,b], P. Borovska[c,d]**, V. Gancheva[c,e]***

[a] *Greek Research and Technology Network, Athens, Greece*
[b] *Scientific Computing Center, Aristotle University of Thessaloniki, Greece*
[c] *National Centre for Supercomputing Applications, Bulgaria*
[d] *Department of Computer Systems, Technical University of Sofia, Bulgaria*
[e] *Department of Programming and Computer Technologies, Technical University of Sofia, Bulgaria*

Abstract

The project focuses on performance investigation and improvement of multiple biological sequence alignment software MSA_BG on the BlueGene/Q supercomputer JUQUEEN. For this purpose, scientific experiments in the area of bioinformatics have been carried out, using as case study influenza virus sequences. The objectives of the project are code optimization, porting, scaling, profiling and performance evaluation of MSA_BG software. To this end we have developed hybrid MPI/OpenMP parallelization on the top of the MPI only code and we showcase the advantages of this approach through the results of benchmark tests that were performed on JUQUEEN. The experimental results show that the hybrid parallel implementation provides considerably better performance than the original code.

Keywords: **Artificial Bee Colony, Bioinformatics, BlueGene/Q, Hybrid Programming, High Performance Computing, Multiple Sequence Alignment, Parallel Programming, Performance**.

1. Introduction

The biological sequence processing is essential for bioinformatics and life science. Scientists are now dependent on databases and access to the biological information. The world DNA databases are accessible for common use and usually contain information for more than one (up to several thousands) individual genomes for each species. Until now 84049 isolates of influenza virus have been sequenced and are available through GenBank[1]. This scientific area requires powerful computing resources for exploring the large sets of biological data, as well as efficient parallel tools for structural genomic and functional analysis. The parallel implementations of methods and algorithms for the analysis of biological data using high-performance computing are essential for accelerating the research.

Multiple sequence alignment (MSA) is an important method for biological sequences analysis and involves more than two biological sequences, generally of the protein, DNA, or RNA type [2]. This method is computationally difficult and is classified as a NP-hard problem [3][4][5].

The innovative parallel algorithm MSA_BG for multiple alignment of biological sequences was proposed as a result of a previous PRACE study [6]. The MSA_BG algorithm is iterative and based on the concept of Artificial Bee Colony metaheuristics and the concept of algorithmic and architectural spaces correlation. The Artificial Bee Colony (ABC) algorithm is an optimization algorithm based on the intelligent foraging behavior of honey bee swarm [7]. In the ABC model, the colony consists of three groups of bees: employed bees, onlookers and scouts. The algorithmic framework of the designed parallel algorithm had already been constructed and the resulting parallel implementation used has been based on MPI only.

* Corresponding e-mail: alcharal@grid.auth.gr
** Corresponding e-mail: pborovska@tu-sofia.bg
*** Corresponding e-mail: vgan@tu-sofia.bg

Within this study we investigate the parallel performance of the MSA_BG algorithm. Optimization is achieved by applying hybrid MPI & OpenMP code development on the initial MPI implementation. The application was ported on the JUQUEEN supercomputer and numerous experiments have been conducted. Profiling and benchmark tests were performed in order to evaluate the performance of the application.

The case study is focused on discovering the evolution of influenza virus and similarity searching between RNA segments of various influenza viruses A strains utilizing all available segments of the influenza virus A on the basis of parallel hybrid program implementation of the MSA_BG multiple sequence alignment method. This will allow identifying the consensus motifs and the variable domains in influenza type A genome.

2. The ABC Algorithm

The Artificial Bee Colony (ABC) algorithm is based on populations. The first step is to generate randomly a partitioned initial population. The colony consists of employed bees, onlookers and scouts. After the initialization, the population repeats the cycle of seeking for food sources. Onlookers and employed bees carry out the exploitation process of food sources, while scouts control the exploration process. These two processes must be carried out together in the search space.

The position of a food source represents a possible solution of the optimization problem and the amount of nectar represents the quality of the proposed solution. A food source is considered exploited and will be abandoned by the bees when a sufficient number of attempts to improve its quality has been reached. A control parameter called “limit” determines the number of times that the bees will try to improve a food source, before they abandon it.

Employed bees are associated with a food source. They carry and share information regarding that particular food source. The number of employed bees is equal to the number of food sources around the hive. An employed bee becomes a scout, as soon as her food source has been exploited and is looking for any food source without any guidance.

A scout modifies the positions of the food sources in his memory and remembers the new position of a food source. In case that a new source nectar amount is greater than the previous, the scout remembers the position of the new source and forgets the old one. Otherwise the scout remembers the position of the previous source in the memory. Once all the scouts complete the search process, they return to the hive and share information about the positions of food sources with onlookers through a dance.

Regarding the exchange of information, an important part of the hive is the dancing area where communication among the bees takes place. This is how collective knowledge is formed. The dance of the bees is called a “waggle dance”.

Each onlooker evaluates the information for the nectar according to the dance of scouts and then selects a food source according to the amount of food in the source. The onlooker compares quantities of nectar in the new source with that already stored. If the amount of the nectar is greater in the new source, the bee remembers the new position and forgets the old one.

For the purposes of MSA_BG application two additional bee roles have been used, which do not exist in the original ABC algorithm: the mother bee of each beehive and the queen bee of the colony. After the cycle of seeking for food sources is complete, the mother bee of a hive will become aware of the best quality solution found in her own hive. Thereafter, each mother bee is responsible to give that solution to the queen bee of the colony. The queen bee determines the elite solution found between all the hives.

3. Multiple Sequence Alignment Algorithm MSA_BG

A new highly scalable and locality aware parallel algorithm MSA_BG for multiple alignment of biological sequences is presented in [8]. MSA_BG is a parallel iterative algorithm with a regular computational and communication system based on data parallelism and replica code, which is executed on all computing nodes. The parallel paradigm is Single Program Multiple Data (SPMD) and data decomposition. The granularity is hybrid - coarse granular computing for each node (multithreaded process) that runs multithreading (fine granular) of the cores within the computing node. In the case of hybrid granularity in order to effectively use the resources of supercomputers it is appropriate to use hybrid parallel implementations. The parallel algorithm is designed according to the methodology for the synthesis of parallel algorithms, which is based on the correlation of the parameters of the algorithmic and architectural spaces. The conceptual model of the MSA_BG method for parallel multiple alignment of biological sequences on the basis of the ABC algorithm is shown in Figure 1.

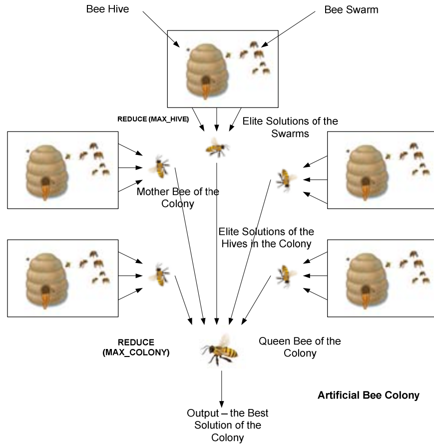


Figure 1: Conceptual model of the MSA_BG method for parallel multiple alignment of biological sequences on the basis of ABC algorithm.

The allocation of computing resources is as follows:

The entire system simulates the behavior of a colony of beehives, and the number of hives is equal to the number of computing nodes. Each computing node simulates the behavior of a hive. Within a hive q swarms are included, where q is the number of segments of the system. The OpenMP threads simulate the behavior of many bees in the swarm. The swarms within a hive work on common lists of best temporary solutions and elite solutions. Each hive has a mother bee, which gets the best quality decisions of all swarms in the hive. The number of mother bees is equal to the number of MPI processes. The queen bee of the colony (MPI process rank 0) finds the elite solution of all the hives in colony.

An overview of the computational algorithm (Figure 1) is the following:

- *Scout 0 reads the sequences from an input file and stores them in the shared memory of the computational node (hive).* Scout bees in the swarms round certain subregions in the searching space and construct a potential solution. Once the scout bees obtain possible (feasible) solution, they return to the hive and begin to dance. The better the quality of a solution generated by a scout is, the higher is possibility to include it in the list of elite solutions. The food source is presented by possible sequence alignments. Scouts generate initial solutions through sequence alignment including gaps. The random generator that is used is based on the Mersenne Twister pseudo random generator that uses a 32-bit word length.
- *Onlookers watch the waggle dances, choose one of the possible solutions and evaluate it.* The quality of obtained solutions is determined by the grade of sequences' similarity. The higher the grade of the solution the better the quality of the obtained alignment, i.e., the criterion of optimality is a maximum similarity score. For the evaluation of the alignment quality, the following method is used:

An assessment by columns is done – in case of nucleotide sequences the numbers of symbols – A, G, C and T are counted. The numbers of symbols are compared and the nucleotides that occur mostly in the different columns are selected. Afterwards, the calculation of assessments in columns is formed, the so-called sequence-favourite (fij), which contains in each position the respective favourite nucleotide in the column (Table 1).

Table 1: Working set - the favourite sequence is marked in red.

A	***G***	***T***	***C***	***A***	***A***	***T***
A	*A*	*T*	*C*	*G*	*A*	*T*
A	*G*	*T*	*C*	*A*	*T*	*T*
A	*G*	-	*G*	*A*	*A*	*G*

Table 2: Scoring Matrix S - The scoring column of the counters is marked in red and consists of similarity scores for each sequence and the sequence-favourite.

1	-1	1	1	-1	1	1	**3**
1	1	1	1	1	-1	1	**6**
1	1	0	-1	1	1	-1	**2**

The sequences are compared to the sequence-favourite. The higher is the similarity to the sequence-favourite, the greater is the grade of a sequence. A scoring matrix is built up which stores (in columns) the values of the evaluation function S for sequences (rows in the matrix). For the grade computation of a sequence (row) i in position j (column) the nucleotide a_{ij} and nucleotide–favourite f_{ij} are used:

$$S_{ij} = 0 \text{ in case } a_{ij} = \text{gap}$$
$$S_{ij} = 1 \text{ in case } a_{ij} = f_{ij}$$
$$S_{ij} = -1 \text{ in case } a_{ij} \neq f_{ij}$$

- *The employed bees select one of the solutions and make attempts to improve it based on local search.* An approach for the modification of the aligned working set of sequences is used:

The column with counters of the scoring matrix S is reviewed and the row (sequence) with the lowest counter value is selected (sequence that differs most from the favourite). Using a random generator two indexes are

selected: one for insertion of a gap (INS) and another for deletion of a gap (DEL) from the list of empty positions (DEL ≠ INS). The generated indexes are compared:

- If DEL > INS, then all characters in positions between INS and DEL are shifted one position to the right (shift_right).
- If DEL < INS the characters are shifted one position to the left (shift_left).

After a number of modifications, employed bees suspend the processing of the current working set and write down the best solution in the list of elite solutions that is sorted in descending order. The condition for termination of the parallel algorithm is the number of iterations.

- *Finally, the mother bee shall inform the queen bee of the colony and send her the quality of the best solution (elite solution).* The colony queen through collective communication reduction with the operation MAX gets the quality of the elite decisions by hives' mother bees and determines the best one. Finally the queen bee sends messages to the mother bee holding the best solution to display the resultant sequences alignment.

4. MSA_BG Algorithm in Computational Steps

A description of the algorithmic framework for the parallel multiple sequence alignment in steps follows:

1. Parse input file - Read the sequences

Each MPI process reads the sequences from the input file and calculates the maximum sequence length. If the variety of sequence lengths is less than a limit, a number of variety gaps are added.

2. Alignment of the sequences

The sequences are aligned by adding gaps in random positions, so that their length is equal to the maximum sequence length. For each sequence (row) a dynamic data structure containing the indexes of the gaps in the sequences has been generated. Each MPI process iterates through the entire set of sequences. Therefore, the number of iterations is equal to the number of processes. This is a highly parallel task as there is no dependency between the sequences.

3. The sequence-favourite is created

Every MPI process iterates through each column on the array of the sequences in order to find the favourite nucleotides. The iterations are equal in number to the max sequence length. The calculations are independent for each genome. Therefore this is also a perfect task for parallel multithreaded execution.

4. The grades of the genomes are calculated

Each MPI process calculates the grade of every sequence in the working set. The grade calculation is independent for each sequence. The number of iterations is equal to the number of the sequences.

5. The sequences are sorted in descending order

The scores of the working sets are stored in the list of solutions, which is sorted, in descending order by the total alignment scores. Each MPI process sorts its own working set of sequences.

6. The quality of the solutions is improved

Minor changes are made in the working set and the quality of the modified alignment is evaluated: the new sequence is compared with the sequence-favourite and a grade is calculated as described in step 4. In case of quality improvement, the new solution is accepted and is stored in the list of "best temporary solutions". Otherwise, the new alignment is ignored. This is an iterative process that can be executed in parallel.

The number of iterations, which corresponds to the number of attempts to improve the quality of each sequence alignment, is split as a division over the number of MPI processes. The total number of iterations is given as an input to the program. In this study, the benchmark tests have been performed using 1 million and 10 millions iterations.

7. The total matrix grade is calculated

After the process of working set improvement is completed, the matrix grade is calculated as the sum of each sequence grade and is stored along with the process rank. An MPI collective communication through reduction

and operation MAX is performed in order to locate the best matrix grade, as well as the process that found the best solution.

8. The best solution is written to the output file

The root process (rank 0) communicates with all MPI processes and informs the process with the best quality solution working set to save the aligned sequences to a file.

5. Experimental Framework

The benchmark tests were run on JUQUEEN, an IBM BlueGene/Q supercomputer. JUQUEEN consists of 28 racks on which 28,672 nodes are installed in total. The processor that is used is IBM PowerPC A2, 1.6 GHz. There are 16 cores per node supporting additional 4-way SMT each. Memory per node is 16 GB. Additionally JUQUEEN hosts 248 I/O nodes [9].

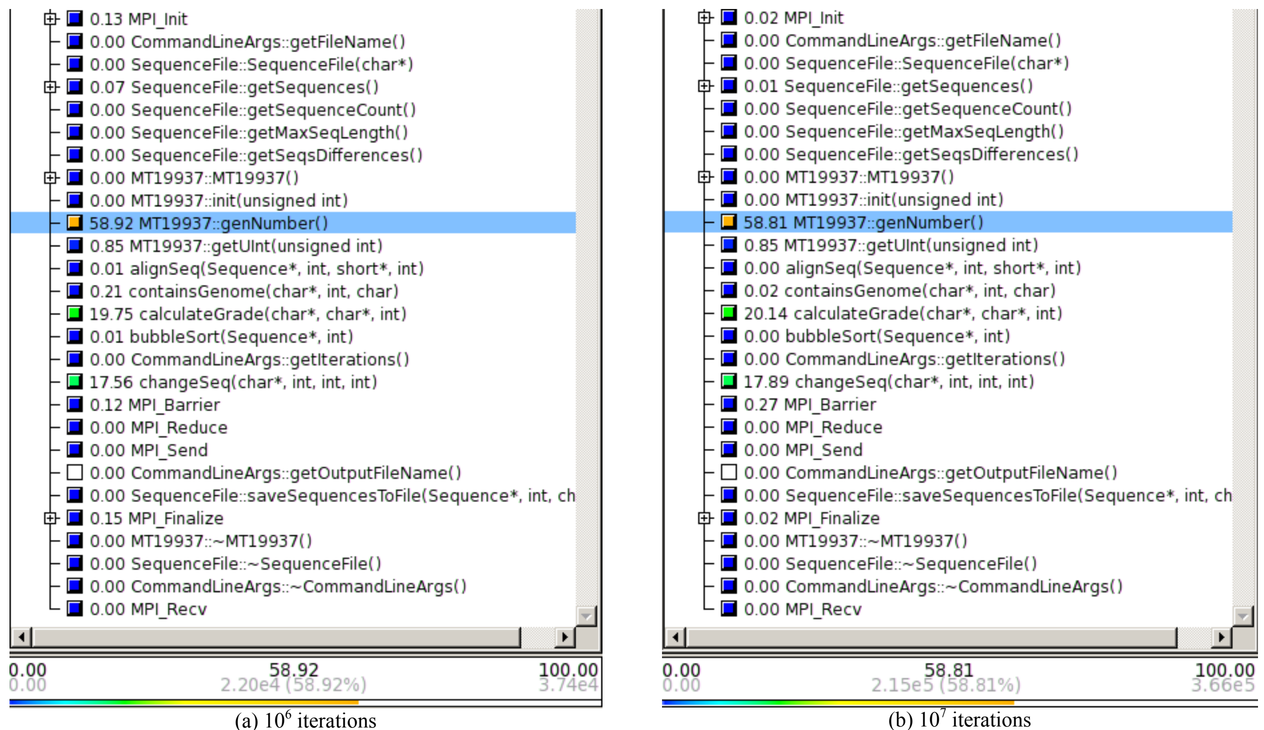
JUQUEEN provides interactive access and submission of batch jobs through two login nodes with processor architecture IBM Power 740. The front-end nodes have identical environments. The operating system is RedHat Linux (6.2) [10].

6. Profiling Results of the MPI only Implementation

A. Scalasca

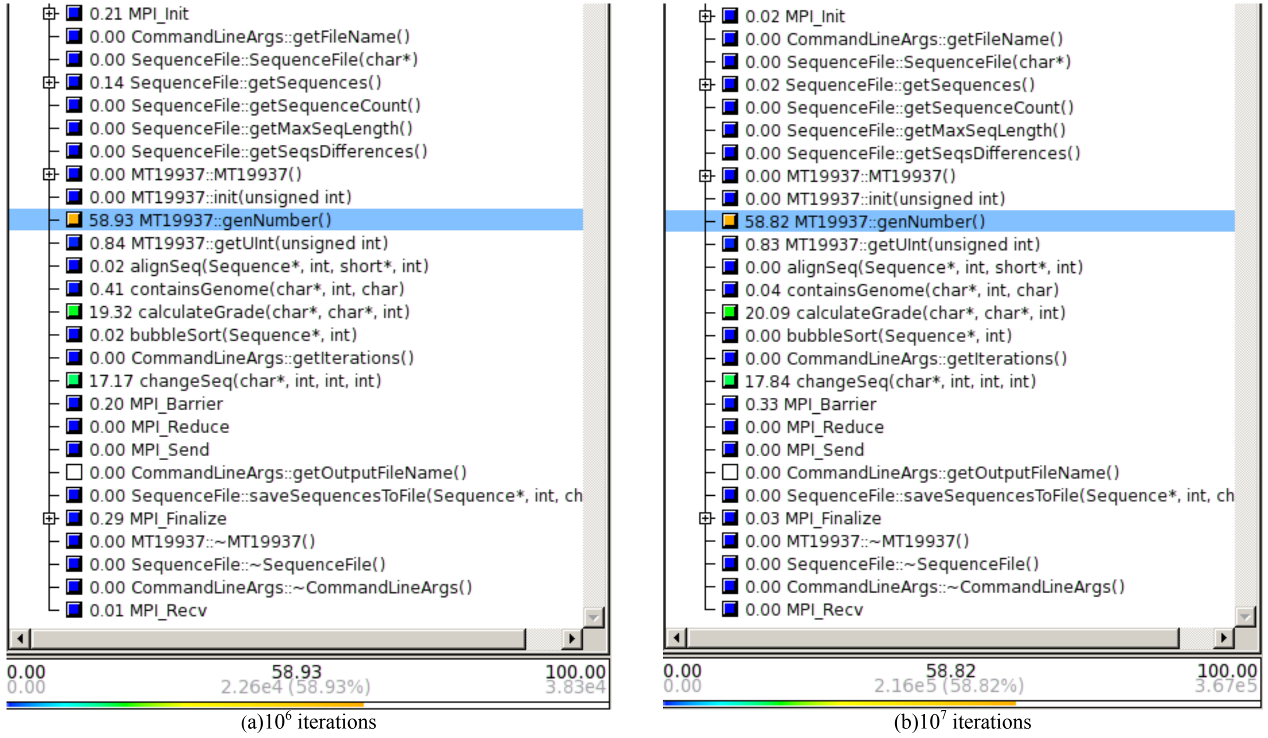
Scalasca has been used for the initial profiling of the code. The Scalasca profiling tool [11] supports performance optimization of parallel programs by measuring and analyzing their runtime behavior.

Tests have shown that the routines that consume the highest amount of time are *genNumber, calculateGrade* and *changeSeq*. *GenNumber* implements the Mersenne Twister pseudo random generator, while *changeSeq* attempts to improve the solution by shifting the retrieved sequence. *CalculateGrade* is used to evaluate the new alignment each time a new modification of the alignment is proposed by *changeSeq*. The total amount of time consumed by these routines is about 95% of the overall execution time. The following screenshots (Figures 2 and 3) from Scalasca document that the rate of time consumption by these 3 routines is very similar when using different numbers of MPI processes and different numbers of iterations as a termination condition.



(a) 10^6 iterations (b) 10^7 iterations

Figure 2: Scalasca profiling results containing execution time percentages using 512 MPI processes on JUQUEEN.



(a)10^6 iterations (b)10^7 iterations

Figure 3: Scalasca profiling results containing execution time percentages using 1024 MPI processes on JUQUEEN.

Important parts of the application, considering I/O, are steps 1 and 8, as described in section 4. At step 1 all MPI processes read the sequences from a common input file. The function *getSequences* is called by every process in order to parse the sequences from the input file and store them in their own memory space as the initial working set. At step 8 the best quality solution is written to the output file. The MPI process which holds the best solution, calls *saveSequencesToFile* to save the final solution.

As a result the total time spent at the *getSequences* calls is proportional to the number of MPI processes, while the time for the *saveSequencesToFile* call stays almost constant for the same problem size.

However, in all cases the percentage of time spent for I/O is below 0.2% of the overall execution time.

B. Valgrind

The profiling tool Callgrind, which is provided through the Valgrind tool suite [12], was used. Callgrind records the call history among functions in a program's run as a call-graph. The collected data consists of the number of instructions executed, their relationship to source lines, the caller/callee relationship between functions, and the numbers of such calls. For graphical visualization of data the KCachegrind tool was used [13].

In this case the analysis is done separately for each MPI process. The Figures 4 and 5 display screenshots taken from KCachegrind corresponding to runs of the program using 4 MPI processes and 10^6 iterations.

In Figure 4 an analysis of the most time consuming part of the source code is displayed which corresponds to the step of the program, which improves the quality of the solutions (step 7). The number of calls for *changeSeq* and *calculateGrade* is equal to number of sequences times the number of iterations and divided with the total number of processes. From the analysis it is obvious that *genNumber* is called twice per iteration in order to generate DEL and INS positions.

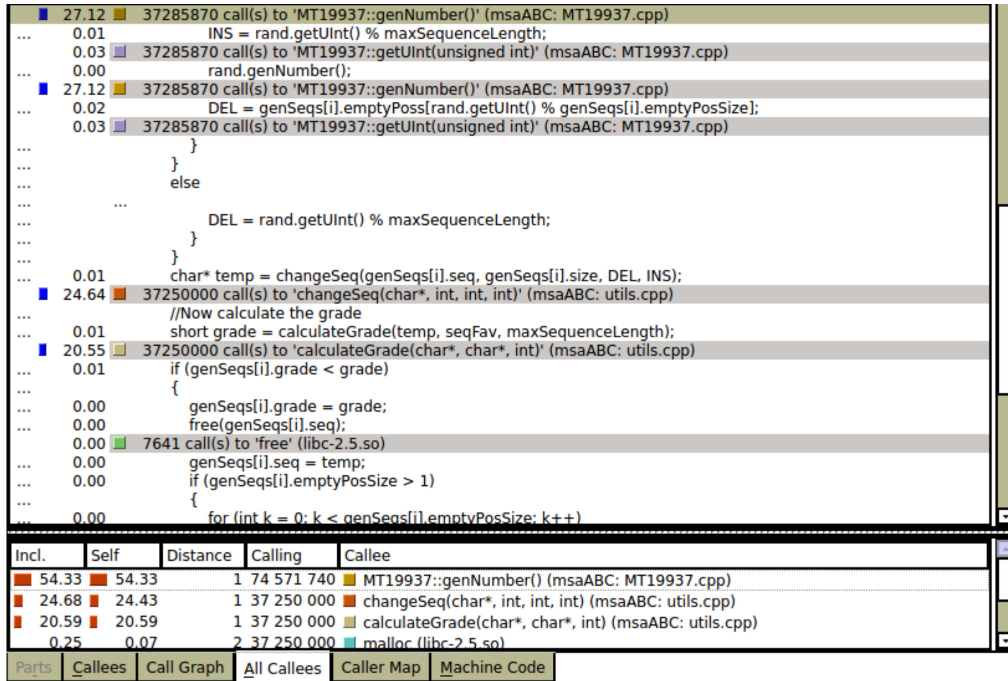


Figure 4: KCachegrind screenshot. The *genNumber* function is called twice per iteration.

Figure 5 presents the cycle estimation percentage for *genNumber*, *changeSeq* and *calculateGrade* over the total number of cycles, which occur independently for a single MPI process.

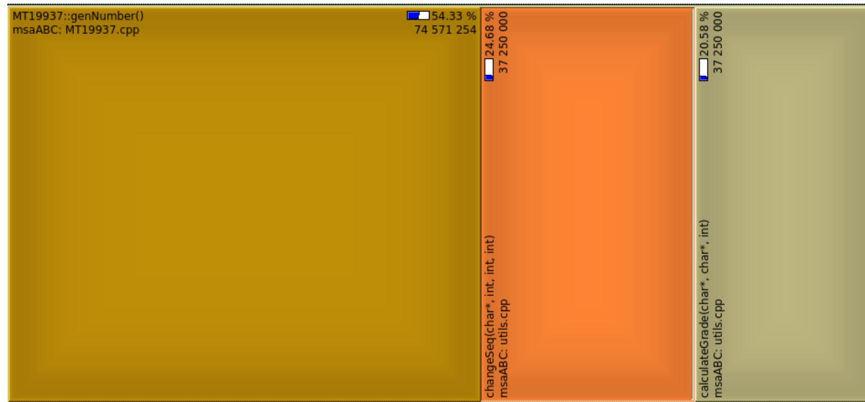


Figure 5: KCachegrind screenshot. The estimation of the cycles percentage and the total number of calls to the 3 routines *genNumber*, *changeSeq* and *calculateGrade* per MPI process is displayed.

Within the following Table we present the minimum and maximum percentages over the total wall time for the steps of the program as described in section 4. These minimum and maximum percentages are computed from several MPI benchmarks using 512, 1024, 2048, 4096, 8192 and 16384 MPI processes.

Table 3: Minimum and maximum percentage of execution times per algorithmic step (section 4).

Step	1	2	3	4	5	6	7	8
MIN	0.0098	0.1408	0.0094	0.0011	0.0009	68.0117	0.0001	0.0134
MAX	2.0198	3.5234	0.2429	0.0280	0.0276	99.3926	0.0020	0.3717

7. Hybrid MPI/OpenMP Implementation

The concept of hybrid parallelism that was implemented on MSA_BG is based on simple logic. Each MPI process forks multiple OpenMP threads to work in parallel with the sequences in the working set. The parallel hybrid MPI/OpenMP computational model of MSA_BG algorithm for multiple sequence alignment is presented in Figure 6.

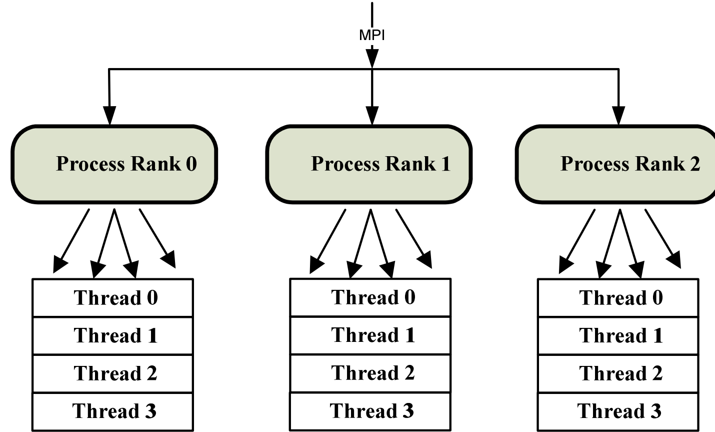


Figure 6: Hybrid OpenMP/MPI parallel computational model of MSA_BG algorithm.

A Hybrid MPI/OpenMP version of the code has been implemented in order to exploit more efficiently the whole shared/distributed memory hierarchy of the JUQUEEN system. The software is written entirely in C++ without any external dependencies to third party libraries.

The steps 2, 3, 4 and 6 of the algorithm described in section 4 are performed in iterations until every sequence is processed. In each **For Loop** a parallel for OpenMP directive is placed. This results in having each OpenMP thread assigned to a different sequence every time until the work that needs to be done is completed.

Initial benchmark tests have shown that increasing the number of threads for the same number of MPI processes and nodes decreases the execution time significantly. Within the next Table we have measured the execution wall time of step 6 (the most time consuming part of the code – see section 4) when adding 2, 4, 8 ad 16 OpenMP threads per MPI process. We also measure the relative speedup (normalized to using only 2 OpenMP threads per MPI process). These results correspond to using 512 processes allocating 4 MPI tasks per JUQUEEN node and for a total of 10^7 iterations.

Table 4: Relative speedup of execution time of step 6 when using OpenMP directives.

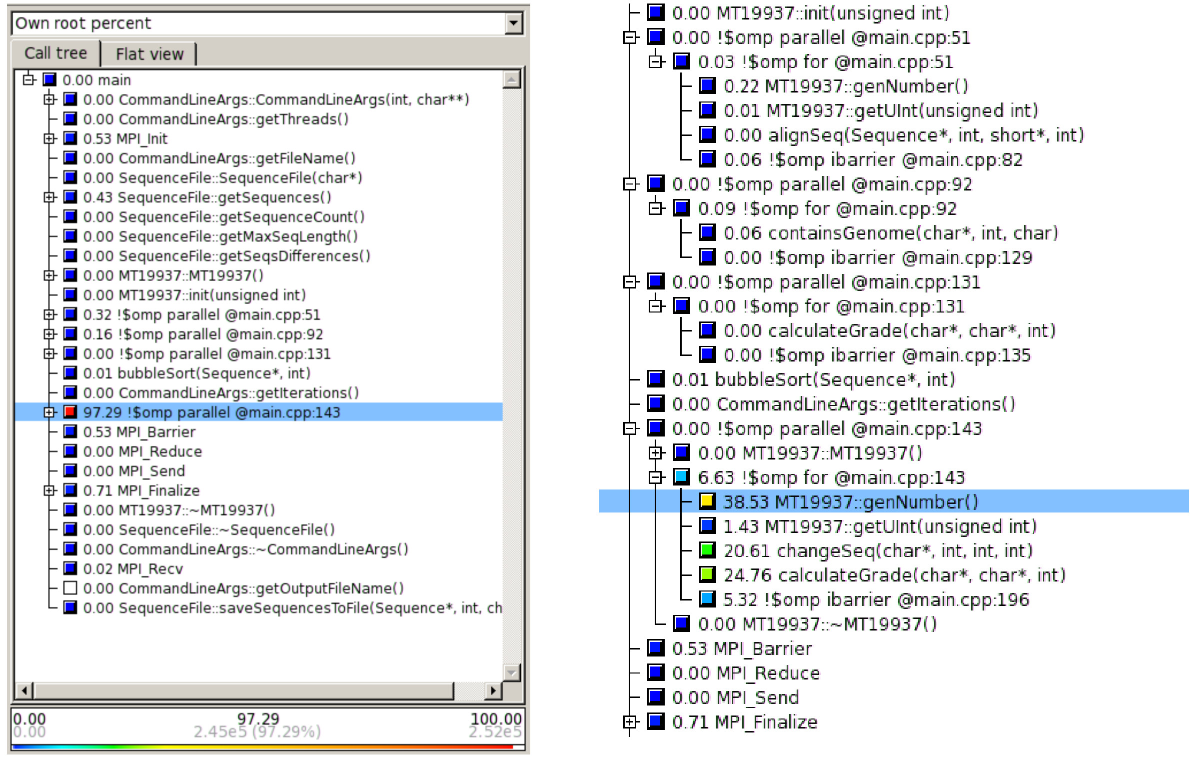
#OpenMP threads	walltime for step 6 (seconds)	relative speedup
2	241.33	1.00
4	122.65	1.96
8	34.04	7.08
16	27.57	8.75

Different implementations of hybrid parallelism that were tested, have not resulted in performance improvement, so they will not be discussed further in this study.

8. Profiling of the Hybrid Implementation with Scalasca

The initial MPI implementation was modified by including OpenMP directives in the source code of the MSA_BG application in steps 2, 3, 4 and 6 as described in section 7.

Figure 7 includes two screenshots taken from the Scalasca profiling tool which correspond to the profiling of the hybrid MPI/OpenMP implementation. The 6th step of the algorithm that improves the sequences alignments consumes almost the most significant part of the total execution time. In Figure 7(b) the four OpenMP parallel regions are displayed in detail. The routines *genNumber*, *changeSeq* and *calculateGrade* still consume a large proportion of the overall execution time. However, in this case they are executed in parallel by a number of OpenMP threads. This results in a significant decrease of the total execution time.



(a) Call tree - cpu time percentages.

(b) OpenMP parallel regions in detail.

Figure 7: Scalasca screenshots corresponding to 1024 MPI processes with 16 OpenMP threads per process. The problem size corresponds to 10^7 iterations.

9. Considerations on OpenMP Scheduling

At run time, the iterations that are assigned to OpenMP threads depend on the chunk size used and the scheduling type. The *schedule* OpenMP directive clause allows specification of the chunking method for parallelization in a work sharing construct, as a for-loop. If the determined type of scheduling is static, the chunks will contain a number of iterations that is equal to the total number of iterations divided by the number of threads. Each thread will be assigned one of these chunks. In case dynamic scheduling is chosen, threads are assigned to a particular number of iterations (default chunk size is 1). Once a thread finishes the work that it has been assigned with it receives another chunk to process.

For the steps 2, 3 and 4 of MSA_BG algorithm, dynamic scheduling was preferred, as it results constantly in a slightly greater decrease of total execution time. This is shown also in Table 5.

Table 5: Comparison of static and dynamic OpenMP scheduling clauses for steps 2, 3 and 4. The numbers in the cells correspond to the percentile decrease of execution time when adding the OpenMP layer to the MPI only code (higher is better).

step	*2*	*3*	*4*
static	92.77	91.87	93.38
dynamic	93.93	95.63	94.16

However, the usage of static scheduling for step 6 in general presented slightly better results throughout all benchmark tests. Thus, only for the 6th step of the algorithm the static clause has been preferred.

10. MPI/OpenMP Configuration and Compilation

The BlueGene/Q system has a distributed memory system and uses explicit message passing to communicate between tasks that are running on different nodes. Within each node a common (shared) memory of 16 GB may be used for threading. The Blue Gene/Q system supports shared-memory parallelism on single nodes [14].

The BlueGene/Q MPI implementation is based on the MPICH2 standard [15] and uses the IBM Parallel Active Messaging Interface (PAMI) as a low-level messaging interface. The BlueGene/Q PAMI implementation directly accesses the BlueGene/Q hardware through the message unit system programming interface (MUSPI).

The OpenMP API for shared-memory parallel programming in C/C++ and Fortran is supported by the IBM extensible language (XL) compilers and the GNU GCC compilers on the Blue Gene/Q system. The IBM XL compilers provide support for OpenMP v3.1. The GNU compilers provide support for OpenMP v3.0.

The IBM XL MPI compiler wrapper for C++ has been used with the xl version of MPICH, PAMI and MUSPI libraries. Other compilation options that were tested on JUQUEEN for MSA_BG did not result in performance improvement. For the OpenMP implementation the thread safe version of the compiler was used [16] with the flags **`-qsmp=omp`** and **`-qnosave`**. Additionally, compiler options **`-O3 -qstrict -qarch=qp -qtune=qp`** were used for aggressive optimization with no impact on accuracy. Higher optimization levels did not result in further performance improvement.

11. Execution Modes

The smallest allocation unit on the JUQUEEN system is 32 compute nodes (512 processor cores) and the maximum number of ranks per node is 64. The number of tasks (ranks per node) needs to be chosen as a power of 2.

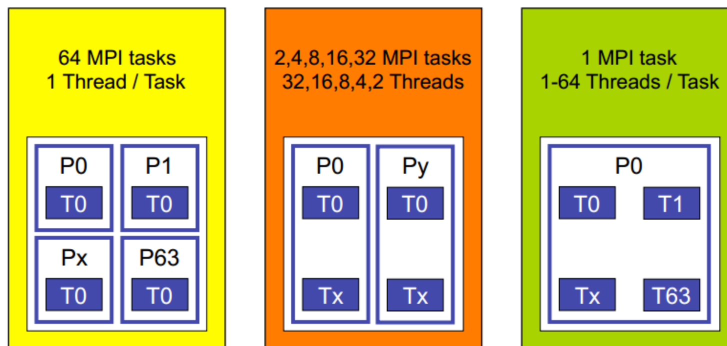


Figure 8: Execution modes in BG/Q [15].

According to our initial results presented in section 7 the performance of the overall code is significantly improved when the number of OpenMP threads is increased. Therefore, the set up that was used for benchmark tests using the hybrid MPI/OpenMP implementation has been chosen in accordance to the configurations listed in Table 6.

Table 6: Configuration alternatives for benchmarking the hybrid MPI/OpenMP implementation.

Configuration Number	#Ranks per Node	#OpenMP Threads
1	32	2
2	16	4
3	8	8
4	4	16

12. Benchmark Tests

Benchmark tests have been conducted on the JUQUEEN supercomputer in order to measure and tune the performance of the application. Similarity searching between RNA segments of influenza viruses sequence has been carried out based on the parallel MPI only version and the hybrid MPI/OpenMP version of MSA_BG software. The experiments that are presented use various numbers of computing nodes, MPI processes and hybrid MPI/OpenMP configurations. The conditions of termination that were used are 10^6 and 10^7 iterations which refer to attempts for improvement of each sequence alignment quality. The input file that was used for the benchmark tests contains 149 sequences with a maximum length of 1036 nucleotides.

A. MPI only Implementation

The results retrieved from the benchmark tests on the implementation which uses only MPI, show that the application scales well as the number of MPI processes increases (Figure 9). The most effective configuration is allocating 16 tasks per node (thus using 16 cores per node without SMT). However, it should be noted that using less tasks per computing node results in allocating a greater number of nodes. Reducing further the number of ranks per node does not result in better performance.

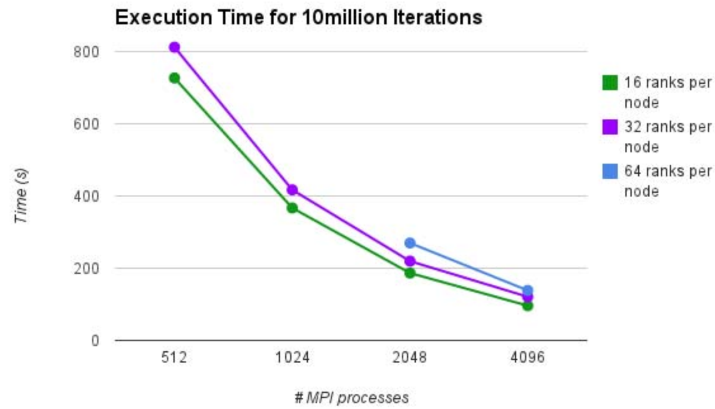


Figure 9: Execution times of the MPI only implementation with varying numbers of MPI processes and BG/Q nodes.

B. Hybrid MPI/OpenMP Parallelization

In this subsection the final results from experimental simulations using hybrid MPI/OpenMP implementation of MSA_BG are presented.

Tables 7 and 8 present the speedup achieved when allocating a constant number of nodes with a different set of MPI ranks per node and OpenMP threads. The speedup is normalized to the MPI only version corresponding wall time for the same number of nodes, using 64 MPI tasks per node (which is the maximum number of ranks per node).

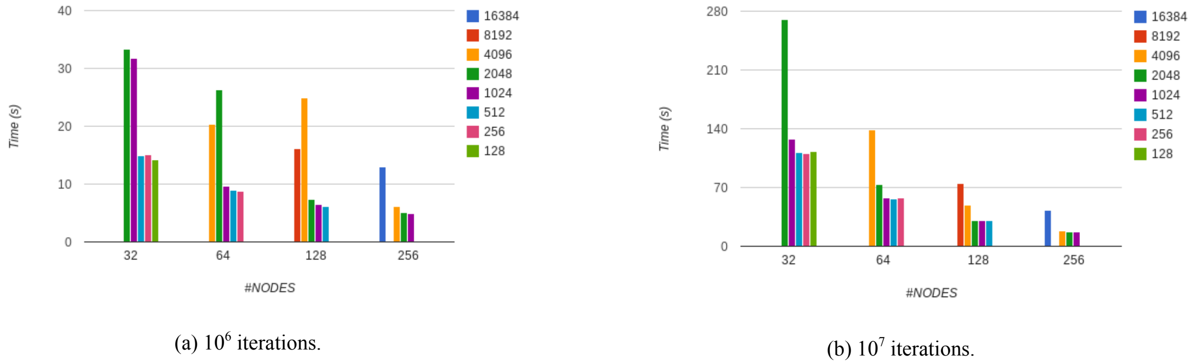
Table 7: Relative speedup of hybrid implementation for problem size of 10^6 iterations.

#NODES	#MPI processes	#OMP threads	Wall time (seconds)	relative speedup
	2048	-	33.25	1.00
	1024	2	31.69	1.05
32	512	4	14.91	2.23
	256	8	15.06	2.21
	128	16	14.23	2.34
	4096	-	20.30	1.00
	2048	2	26.39	0.77
64	1024	4	9.57	2.12
	512	8	8.89	2.28
	256	16	8.75	2.32
	8192	-	16.13	1.00
	4096	2	24.95	0.65
128	2048	4	7.42	2.18
	1024	8	6.55	2.46
	512	16	6.21	2.60
	16384	-	13.03	1.00
	4096	4	6.06	2.15
256	2048	8	5.14	2.53
	1024	16	4.89	2.66

Table 8: Relative speedup of hybrid implementation for problem size of 10^7 iterations.

#NODES	#MPI processes	#OMP threads	Wall time (seconds)	relative speedup
	2048	-	269.75	1.00
	1024	2	127.22	2.12
32	512	4	111.31	2.42
	256	8	110.58	2.44
	128	16	113.39	2.38
	4096	-	138.44	1.00
	2048	2	74.12	1.87
64	1024	4	57.73	2.40
	512	8	57.09	2.43
	256	16	58.29	2.37
	8192	-	75.23	1.00
	4096	2	48.79	1.54
128	2048	4	31.05	2.42
	1024	8	30.41	2.47
	512	16	30.92	1.38
	16384	-	42.71	1.00
	4096	4	18.18	2.35
256	2048	8	17.34	2.46
	1024	16	17.24	2.48

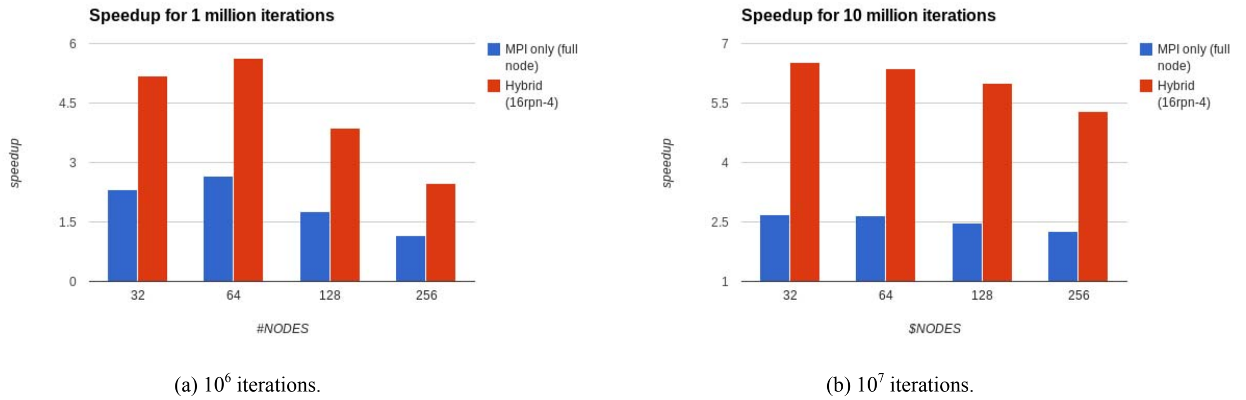
In Figure 10 we have plotted the execution times of our benchmark case for two different problem sizes. The first bar for each number of nodes corresponds to the execution time of the MPI only version of the code when 64 MPI tasks per node are used. The remaining four bars within each nodes number correspond to the execution times of the hybrid version using the configurations that are presented in Table 6 (in the same order). The setup that uses 16 MPI processes per node with 4 OpenMP threads seems to indicate a threshold regarding performance. When increasing the number of threads further for a constant number of nodes, performance may still increase, but not significantly.



(a) 10^6 iterations. (b) 10^7 iterations.

Figure 10: Execution times of the MPI only and the hybrid versions of the code. The setup that uses 16 MPI processes per node with 4 OpenMP threads seems to be a threshold over which performance may still increase, but not significantly.

Figure 11 presents the gain of using the hybrid MPI/OpenMP implementation in comparison to exploiting the full node using only MPI processes. Corresponding runs that use the 16 cores available on each node are used as a base in order to normalize the relative speedup. Therefore, in case a configuration is used that occupies 16 MPI tasks per node, there would be two options. Either to use 4 times more MPI processes or exploiting the remaining tasks on each node with OpenMP threads. Through Figure 11 we document that it is much more preferable to apply the second option using the hybrid implementation as it results in a more significant speedup.



(a) 10^6 iterations. (b) 10^7 iterations.

Figure 11: Relative speedups obtained via MPI only and hybrid MPI/OpenMP implementation.

Figure 12 displays in detail the execution times for different numbers of nodes, MPI processes and OpenMP threads when the condition of termination is 10^7 iterations. The blue line corresponds to runs using the four different hybrid configurations, which are displayed in Table 6 with a fixed number of MPI processes. The red line is used as a comparison and refers to using only MPI processes, with 64 ranks per node (full node).

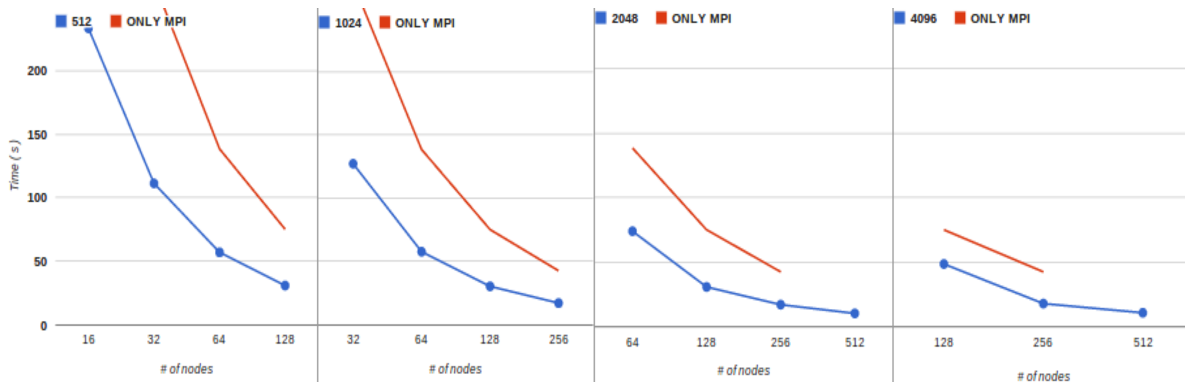


Figure 12: Execution times for different numbers of nodes, MPI processes and OpenMP threads for 10^7 iterations.

13. Parallel Performance Evaluation of Hybrid Implementation and Results Analysis

Similarity searching between RNA segments of various influenza viruses A/H1N1 strains obtained from Genbank [1] has been carried out based on the hybrid MPI/OpenMP version of MSA_BG algorithm for multiple sequence alignment. Some experiments using various numbers of cores and 1 million iterations have been conducted. The experimental results in Table 9, Figure 13 and Figure 14 show that the parallel program implementation for multiple sequence alignment scales well as the number of the cores increases.

Table 9: Execution time and acceleration of MSA_BG algorithm using various numbers of cores and various sequences.

Host, subtype	Segment number	Length of seq.	Number of seq.	Input file (MB)	Execution time, minutes			Acceleration, %	
					128 cores	*256 cores*	*512 cores*	*256 cores*	*512 cores*
All hosts, subtype H1N1	All H1N1 #1	2340	3780	8,78	11,33	6,51	3,85	42,54	66,02
	All H1N1 #4	1750	5992	10,52	14,58	8,45	4,67	42,04	67,97
	All H1N1 #6	1460	6128	8,90	13,35	7,82	4,12	41,42	69,14
	All H1N1 #8	890	3996	3,58	7,12	4,14	2,16	41,85	69,66
Human, all subtypes	Human #1	2340	5850	13,61	17,2	10,06	5,64	41,51	67,21
	Human #4	1750	8999	15,77	22,01	12,47	7,23	43,34	67,15
	Human #6	1460	9662	14,05	20,55	12,04	6,45	41,41	68,61
	Human #8	890	6150	5,50	10,26	6,1	3,42	40,55	66,67

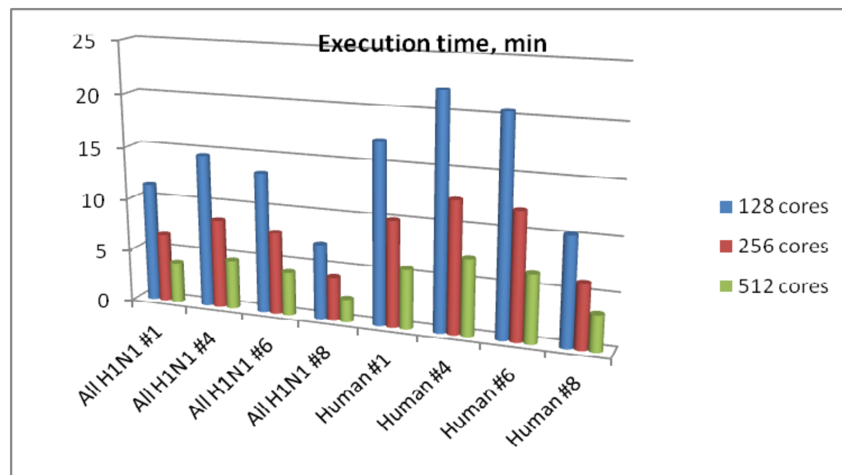


Figure 13: Execution time of MSA_BG algorithm as a function of number of cores.

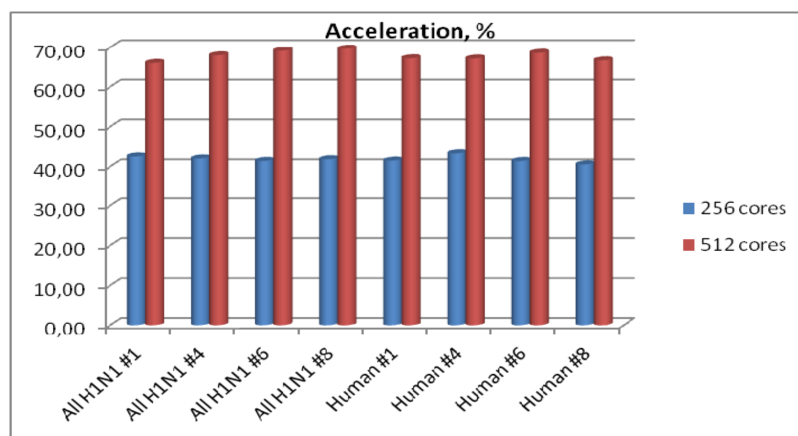


Figure 14: Acceleration of MSA_BG algorithm with respect to 128 cores.

The molecular biology outcome of the experiments is that the consensus motifs and the variable domains in Influenza virus A have been determined and published using the Unipro UGENE editor [17]. The results are presented in Figure 15.

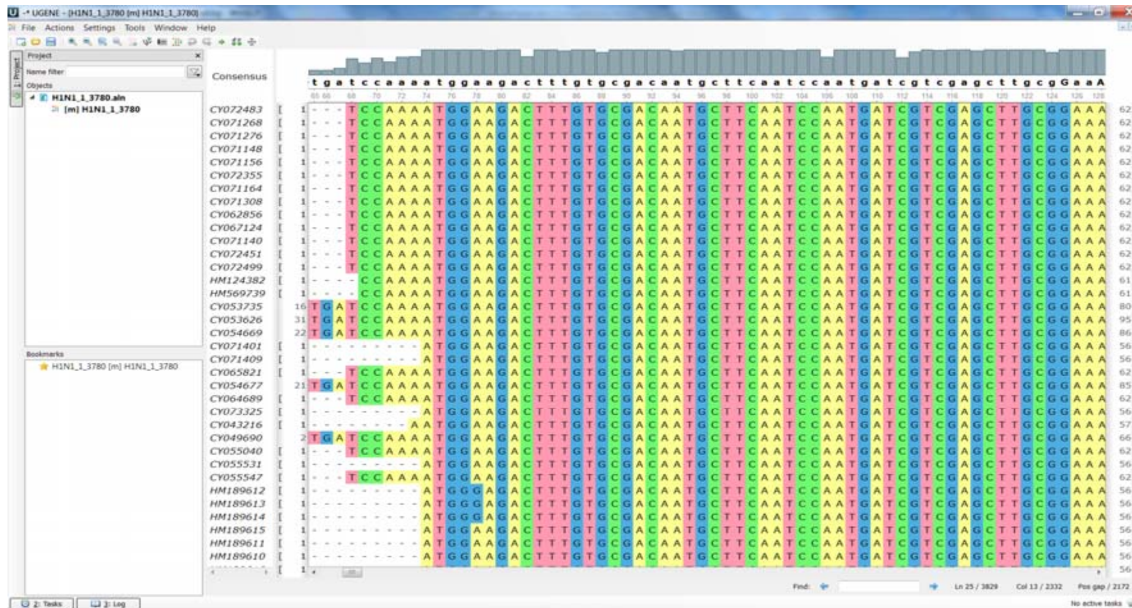


Figure 15: Finding out consensus and variable domains in the case of Human Influenza Virus A/H1N1; output by graphic editor Unipro UGENE.

14. Conclusions

The parallel software MSA_BG for multiple sequence alignment has been ported and tuned on the Blue Gene/Q supercomputer JUQUEEN. Hybrid MPI/OpenMP parallelization was implemented and evaluated experimentally. Parallel performance was investigated and optimized through benchmark tests and profiling.

The implementation of hybrid MPI/OpenMP parallelization on MSA_BG reduces up to a good factor the overall runtime of the MPI only version of the application as it allows us to fully occupy the cpu capacity of a JUQUEEN node with threads. It should also be noted that runs using the hybrid implementation resulted in better quality of sequence alignments due to additional randomness that was produced by the Mersenne Twister generator.

The performance estimation and analyses show that the hybrid parallel program implementation of MSA_BG algorithm scales well as the number of the cores increases and is well balanced both in respect to the workload and machine size. The optimized code is universal and can be applied for other similar research projects and experiments in the field of bioinformatics. MSA_BG software allows researchers to conduct their experiments and perform simulations with very large amounts of data.

References

[1] GenBank, http://www.ncbi.nlm.nih.gov/Genbank/, retrieved: 20.06.2013.

[2] H. Carrillo and D. Lipman, “The Multiple Sequence Alignment Problem in Biology,” SIAM Journal of Applied Mathematics, vol. 48, no. 5, 1988, pp. 1073-1082.

[3] L. Wang and T. Jiang, “On the complexity of multiple sequence alignment”, Journal of Computational Biology, vol. 1, no. 4, 1994, pp. 337–348, doi:10.1089/cmb.1994.1.337.

[4] W. Just, “Computational complexity of multiple sequence alignment with SP-score”, Journal of Computational Biology, vol. 8, no. 6, 2001, pp. 615–23.

[5] S. Sze, Y. Lu, and Q. Yang, “A polynomial time solvable formulation of multiple sequence alignment”, Journal of Computational Biology, vol. 13, no. 2, 2006, pp. 309–319, doi:10.1089/cmb.2006.13.309.
[6] P. Borovska, V. Gancheva, “Massively Parallel Algorithm for Multiple Sequence Alignment Based on Artificial Bee Colony”, PRACE Whitepaper, 2013, http://www.prace-ri.eu/IMG/pdf/wp114.pdf
[7] D. Karaboga, “An Idea Based On Honey Bee Swarm for Numerical Optimization,” Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005, http://mf.erciyes.edu.tr/abc/pub/tr06_2005.pdf
[8] P. Borovska, V. Gancheva, N. Landzhev, “Massively Parallel Algorithm for Multiple Biological Sequences Alignment”, 36th International Conference on Telecommunications and Signal Processing (TSP), 2-4 July, 2013, Rome, Italy, pp. 638 - 642.
[9] JUQUEEN – Configuration, http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/Configuration/Configuration_node.html
[10] JUQUEEN – Logging on to JUQUEEN, http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/UserInfo/LogonJuqueen.html.
[11] Scalasca, http://www.scalasca.org/
[12] Valgrind Documentation, Free Software Foundation, 2013, http://valgrind.org/docs/manual/valgrind_manual.pdf.
[13] J. Weidendorfer, F. Zenith, “The KCachegrind Handbook”, http://docs.kde.org/stable/en/kdesdk/kcachegrind/kcachegrind.pdf.
[14] Megan Gilge, “IBM System Blue Gene Solution: Blue Gene/Q Application Development”, IBM international Tecnical Support Organization, 2013, http://www.redbooks.ibm.com/redbooks/pdfs/sg247948.pdf.
[15] M. Stephan, “JUQUEEN: Blue Gene/Q - System Architecture”, Forschungszentrum Jülich http://www.training.prace-ri.eu/uploads/tx_pracetmo/JUQUEENSystemArchitecture.pdf.
[16] Florian Janetzko, “JUQUEEN: Application Stack and Best Practices,” Jülich Supercomputing Centre, Institute for Advanced Simulation, Forschungszentrum Jülich, http://www.training.prace-ri.eu/uploads/tx_pracetmo/JUQUEENAppStackBestPractises.pdf
[17] Unipro UGENE: Integrated Bioinformatics Tools, http://ugene.unipro.ru/

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement nos. RI-283493 and RI-312763.