The Journal of Supercomputing (2019) 75:3499–3520
https://doi.org/10.1007/s11227-019-02859-w

# Optimized cloud-based scheduling for protein secondary structure analysis

**Marco Ferretti**[1] · **Luigi Santangelo**[1] · **Mirto Musci**[1]

Published online: 25 April 2019

## Abstract

In the domain of proteomics, an in-depth analysis of the 3D structure of a protein is of paramount importance for many biological studies and applications. At the secondary level, protein structure can be described in terms of motifs, recurrent patterns of smaller biological structures called secondary structure elements. In this paper, the focus is on the identification of geometrical motifs in different proteins using the Cross Motif Search (CMS) algorithm. Such task, due to the high computational cost of CMS with respect to traditional alignment algorithms, is very demanding, and thus, parallel processing is mandatory. In previous papers, CMS parallelization has been already studied from the HPC standpoint. Since cloud computing is emerging as an alternative to on-premise HPC systems, it is worthwhile examining the feasibility and possible advantages in terms of both performance and costs, of migrating to a cloud implementation. This paper is an extension of a preliminary work carried out on the cloud parallelization of CMS. The paper has two main contributions. First of all, an analytic model of the communication pattern of CMS is described, in order to get insights on the performance of the application when executed on a cloud infrastructure. Secondly, an optimized "location-aware" scheduling policy to assign workload to the application workers is introduced, in order to minimize internode communication in a cloud setting. Experiments are presented in order to validate the newly introduced scheduling policy and assess the performance of the cloud implementation of CMS. The results presented in this paper are general, in the sense that they can be applied to any other algorithm with a communication pattern similar to the one of the target applications.

**Keywords** Proteomics · Cloud computing · HPC · Cross Motif Search · CINECA · Google Cloud · pLogP

✉ Luigi Santangelo
luigi.santangelo@unipv.it

Marco Ferretti
marco.ferretti@unipv.it

[1] Department of Electrical, Computer and Biological Engineering, University of Pavia, Pavia, Italy

## 1 Introduction

Cloud computing is a new paradigm for sharing computational resources in a geographically distributed way, using virtualization layers in order to hide the underlying hardware. Cloud computing is an on-demand service, which does not need human interaction; it is broadly available on the Internet; it allows for composite pools of hardware resources; it can be quickly scaled in or out and it is based on a "pay-as-you-go" cost model. For all of these reasons, cloud computing is a promising environment for running parallel applications, even scientific ones which are usually very expensive in terms of computations and are traditionally executed on HPC systems [2, 3].

The main difference between cloud and HPC is that HPC systems provide a more direct contact to the bare metal (i.e., the underlying physical hardware) which instead is hidden within a cloud solution and thus allow for better performance optimization. However, cloud computing dynamic scaling and broad availability can bridge and even overcome this gap.

According to the literature, migrating an existing HPC application to the cloud could be beneficial in terms of performance, costs or other factors. However, several migration failures have also been reported, mostly due to the high volume of communication of the target application [4–9]. Generally speaking, neither infrastructure is inherently better than the other, and an informed choice must be made, on a case-by-case instance.

This work focuses on the migration of the Cross Motif Search (CMS) application to the cloud, and it is an extension of a preliminary work presented at the PBIO 2018 conference [1]. CMS is a proteomics application (see Sect. 2), with the focus on geometrical motif identification, which the authors have already extensively analyzed in its parallel implementations for on-premise infrastructures [10–16]. In particular, we report on an advanced modeling effort designed to characterize the master/worker communication protocol of CMS for cloud deployment. The model will be used to asses the feasibility of the migration and to design an optimization to reduce the impact of communication on performance.

With respect to the preliminary work [1], the present contribution introduces a new scheduling policy which can significantly reduce internode communication while keeping good load balancing, and then compares the global execution of Cross Motif Search in HPC and in Cloud systems, giving a measure of the application's scalability and speedup.

### 1.1 Related works

In the past, researchers have proposed different methodologies to address load balancing issues. A good taxonomy of all load balancing strategies can be found in [17–20]. Many of these algorithms have also been compared and optimized for cloud infrastructure [21–23]. The authors of [24] present a model that shows how heterogeneity in the cloud can result in a performance degradation. In [25], instead, authors present a middleware technology which implements load balancing and reduces communication cost. Dynamic load balancing methods for HPC proteomics applications are described in [26, 27]. A very interesting related work is the one presented by Mrozek at al. [28] which develops a highly scalable cloud-based system, named Cloud4Psi, to support researchers in the cloud execution of several protein alignment tools. However, Cloud4Psi is mostly concerned in using the cloud as a service, rather than as a computing infrastructure.

The authors have already performed a similar extensive analysis on a different bioinformatics application, namely BloodFlow, used by surgeons and doctors to model and simulate the global hemodynamic phenomena [29–32]. In this case, the migration to the cloud was proven to be detrimental: The main reason is that BloodFlow is based on a communication pattern which is much more complex than CMS. Generally speaking, many parallel applications are based on similar communication patterns as the one exposed by either CMS and BloodFlow, and thus, we believe that our results can be considered as representative for a wide range of applications.

### 1.2 Structure of the paper

The paper is structured as follows. Section 2 describes the Cross Motif Search algorithm and its HPC implementation and performance. Section 3 identifies all the necessary elements for a cloud migration of CMS, describes the target cloud infrastructure and builds the communication model of CMS in order to assess the feasibility of a cloud implementation. Section 4 describes a new scheduling policy for the workload allocation of CMS, designed to minimize internode communication in a cloud setting. Section 5 presents several experiments that analyze the application scalability on the cloud, validates the communication models presented in Sect. 3 and assesses the load balancing of the application following the introduction of the policy described in Sect. 4. Section 6 discusses the experimental results and concludes the paper.

## 2 Cross Motif Search

### 2.1 Overview and related works

From the biological point of view, looking for similarities in both close and distant evolutionary-related proteins is critical to assess structure–functionality relationships. Comparisons could be made at different levels of the protein structure. For the purpose of this paper, we are only interested in the analysis at the secondary level and in particular in the identification of recurring patterns of secondary structure elements (SSEs) called motifs.

In the literature, many techniques have been studied for motif identification. A comprehensive review is presented in [33]. In this paper, however, our focus is on Cross Motif Search, a radically different algorithm for motif identification with respect to the state of the art.

The novelty of CMS is the focus that it puts on the *geometrical* description of the structural motifs, which could be simply viewed as line segments, rather than on the topological/biological description employed by well-known algorithms such as DALI [34], ProSMoS [35, 36], PROMOTIF [37] or MASS [38].

With respect to other geometrical algorithms, such as secondary structure matching (SSM) [39], CMS is much more precise, as SSM only gives a similarity score for the geometrical structures of pair of proteins, while CMS provides a complete geometrical characterization for each match. The geometrical search performed by CMS is an exhaustive combinatorial process (Sect. 2.2). For this reason, the main shortcoming of CMS with respect to the state of the art is performance, as precise geometrical approaches are inherently slower. Thus, efficient parallel implementations become extremely important (Sect. 2.3).

The relevance of CMS with respect to the state of the art, even with sub-optimal performance, is due to the fact it has been designed with a different goal in mind with respect to other algorithms, that is, to look for *previously unknown* common geometrical structures in "unfamiliar," evolutionary distant proteins (Sect. 2.4). This means that CMS cannot be entirely substituted by well-known and extremely efficient alignment tools.

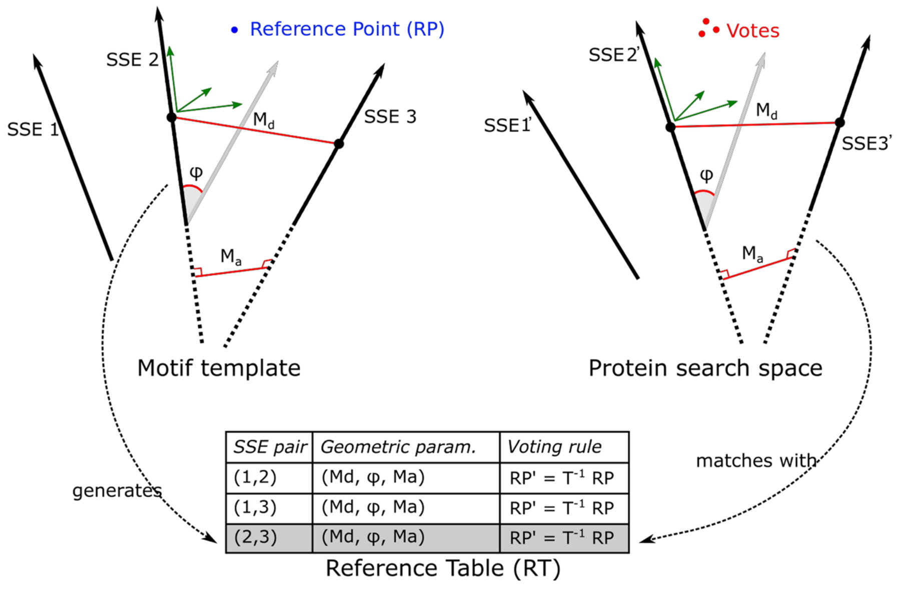
CMS shares a similar goal with [40]. However, even if the authors start from a geometrical approach, they use statistics during the search phase. This approach allows for a great speedup in execution time, but, at the same time, makes them loose the geometrical information that CMS is able to retrieve. The automatic and meaningful analysis of the similarities indentified by CMS is still an open problem (see, for instance, [41–43]). An in-depth analysis of CMS providing an extended bibliography on motif extraction can be found in [15, 16].

## 2.2 The algorithm

As stated above, CMS is based on a simple geometric model of the secondary structure of proteins [15]. In particular, it assumes that each secondary structure element (SSE) can be modeled as a simple line segment in 3D space (Fig. 1) with length, direction and barycenter computed from the constituent amino acids. Consequently, for the purpose of CMS, a motif is a collection of line segment recurring among several proteins.

For simplicity, only the two most common types of SSEs, namely $\alpha$ helices and $\beta$ strands, are considered in the geometric model. The key idea of CMS is to apply computer vision techniques to the geometric model of a protein, and in particular a variant form of the generalized Hough transform [44] to find recurring patterns among different proteins.

The CMS algorithm works exhaustively among a pair of proteins, called, respectively, *source* and *search* proteins, and tries to find each possible geometrical similarity between them. A simplified illustration of the inner workings of CMS is presented in Fig. 1. For each pair of SSEs in the source protein, a selected set of geometrical features are identified, namely the angle between the segment $\phi$, the normal distance between the segments $M_a$ and the distance between the segment



| SSE pair | Geometric param. | Voting rule |
|---|---|---|
| (1,2) | (Md, φ, Ma) | RP' = T$^{-1}$ RP |
| (1,3) | (Md, φ, Ma) | RP' = T$^{-1}$ RP |
| (2,3) | (Md, φ, Ma) | RP' = T$^{-1}$ RP |

**Fig. 1** A simplified illustration of the algorithmic kernel of CMS. For each motif template in a source protein (left), a reference table (RT) is created (bottom) using the distance between barycenter, the normal distance and angle among any pair of SSEs in the template. The RT is used to define a linear transformation that allows to retrieve a single reference point for all the pairs on the left. Finally, all the SSE pairs in a search protein (right) are compared to the RT. A point—called vote—is computed in the search space for each row. A match is found if all votes are close to each other. This is the case in the figure: The set (SSE1, SSE2, SSE3) matches with (SSE1', SSE2' SSE3') and thus is a candidate to be a motif

barycenters $M_d$. The algorithm then groups *each* possible set of SSEs in template motifs, such as the one constituted by SSE1, SSE2 and SSE3 in the figure. This set is not yet a match, but simply a candidate that needs still to be matched. A "reference table" (RT) is assigned to each template. Each possible pair of SSEs in the template contributes a line to the table, containing $M_d$, *phi* and $M_a$. From the table a transformation $T$ is derived, so that for each line in the table, the same reference point (RP) can be identified. In the figure, given that the template is made of three SSEs, the RT has three lines, one each for the pair (SSE1, SSE2), (SSE1, SSE3) and (SSE2, SSE3). The transformation $T$ is then inverted ($T^{-1}$) to define a *voting rule*. Once the RT has been defined, the algorithm moves to the *search* protein. *Each* possible motif template is analyzed in turn. The figure shows the analysis on the search template made of SSE1', SSE2' and SSE3'. The voting rule is then applied to each pair in the search template order to define a point (called *vote*) in the 3D space. According to the property of the Hough transform, if *all* the votes in the search template are close enough to each other, then the current template is similar to the one that generated the reference table. In the figure, as the two templates are similar to each other, the votes will be clustered together; thus, we can conclude that (SSE1, SSE2, SSE3) is geometrically similar to (SSE1', SSE2', SSE3'). Repeating this entire process for each source template and for each search template yields the complete characterization of the geometrical similarities between the source and the search protein, which

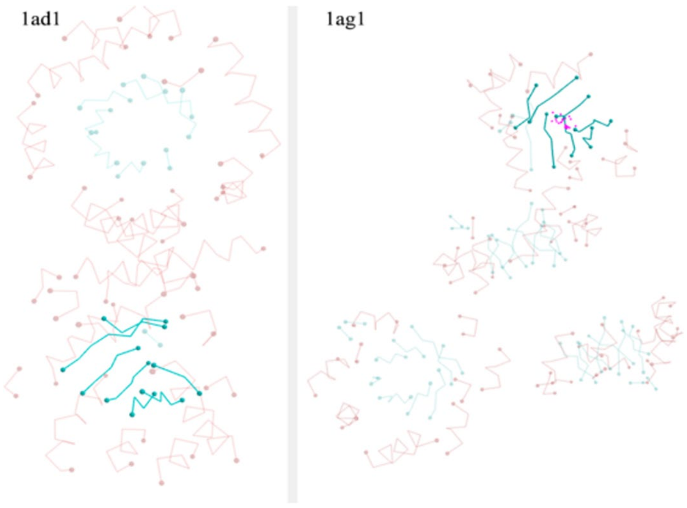


**Fig. 2** A snapshot of the MotifVisualizer application that allows to visualize the output of a CMS run on a pair of proteins (1ad1 and 1ag1 in the figure). $\alpha$ helices are highlighted in red, and $\beta$ strands are highlighted in blue. Among the many similarities that CMS has identified between the two proteins, only an exemplary one is shown. Note that even if CMS is optimized to work on evolutionary distant proteins, the example in figure shows that CMS works also with proteins with high alignment scores (56% according to SSM [39])

also includes the exact position of each match, and all of its biological and geometrical features. The authors have also developed a tool[1] to analyze a given pair of proteins in real time and graphically visualize all the similarities found by CMS. The tool, named MotifVisualizer, is shown in Fig. 2 [45].

## 2.3 HPC implementation

Being based on the exhaustive combinatorial approach described in the previous section, it is clear that CMS is a very computational intensive algorithm, even for a single pair of proteins. Furthermore, in order to characterize a given dataset of proteins, the entire CMS algorithm *kernel* described in the previous section must be repeated all-to-all for each protein pair in the set. Thus, a massive parallel implementation is needed. A hybrid OpenMP-MPI HPC implementation of CMS was described in [10] in order to execute the application on a distributed and shared memory system.

Given a dataset containing $m$ protein files, the hybrid CMS parallel implementation executes a number of kernels equal to:

$$K = \frac{m \times (m-1)}{2} \tag{1}$$

[1] https://vision.unipv.it/bioinformatics/contents/tools.php.

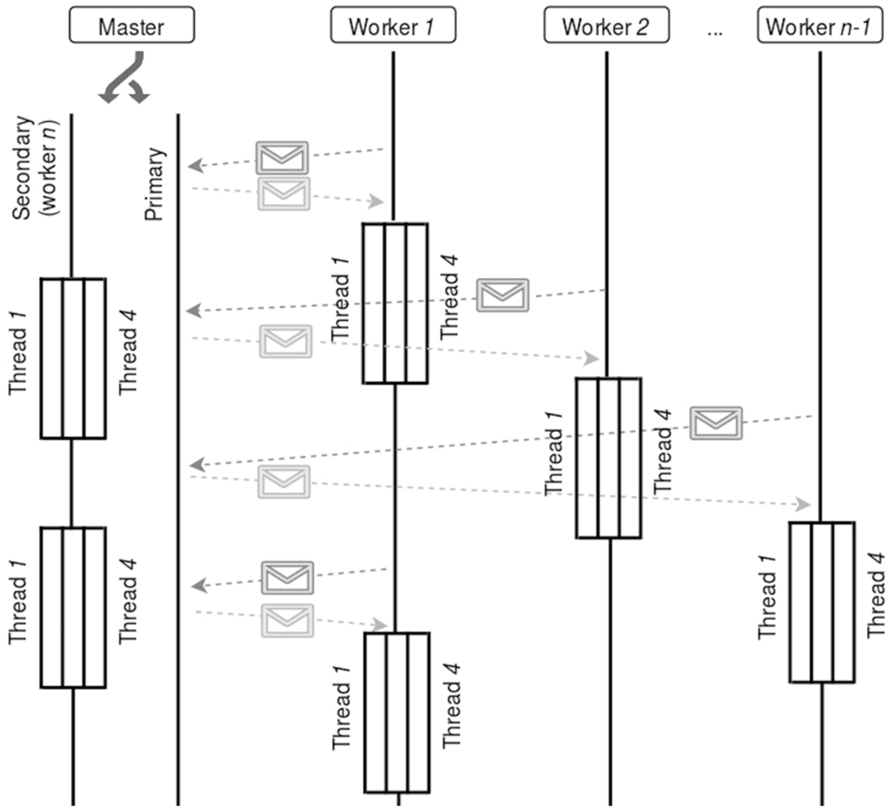


**Fig. 3** The master–workers communication pattern of the HPC implementation of CMS. There are a total of $N$ workers, one running concurrently on the master, the other $N - 1$ running on different distributed tasks, each running an OpenMP instance of the CMS kernel (using four threads in the figure). Workers send a 4-byte message (dark gray envelope) when they are ready to process a new task; the master sends back a 6-KByte reply message (light gray envelope) when work is available. The reply message contains the name of the proteins to be retrieved from a centralized dataset and then processed by the worker

In the implementation, CMS kernels on single pair of proteins are modeled as tasks and are executed concurrently by all the MPI processes in order to complete the whole job, that is, the CMS analysis of an entire dataset of proteins. At the application outset, two different sets of processes are created. The first set contains just one master process; the other set contains $n - 1$ worker processes. Master and workers exchange information by using point-to-point MPI functions, such as MPI_Send and MPI_Recv.

The communication pattern of the CMS HPC implementation is quite trivial, as it is the case for many parallel bioinformatics applications. As shown in Fig. 3, indeed, only data exchanges between master and workers are allowed. No interworker communication happens during the whole execution, and no collective operation is used to gather and scatter data. However, load balancing is non-trivial, as it will become clearer later (see in particular Sect. 4).

With reference to the figure, when a worker is ready, it sends a 4-byte message to the master asking for the next task (i.e., which CMS kernel) to compute. When the master receives the message, it sends back to the worker a 6-Kbyte message containing the name of a yet unprocessed protein pair. When the worker gets the reply message, it starts an OpenMP parallel computation of the CMS kernel on the two proteins. After completing the task, the worker asks again for a new task and the cycle begins anew. When all protein pairs have been exhausted, the master broadcasts an EOF message to all workers, which thus complete their execution. The load balancer algorithm used by the master process can be classified as dynamic, synchronous, demand-driven, centralized and one-time assignment. More details can be found in [11].

In order to further improve performance, the master is implemented in multi-threaded mode too. After starting the application, the master process spawns into two concurrent OpenMP threads. The first one, named primary master, waits for the requests coming from the workers; the secondary master instead behaves as any other worker, but differently from such worker processes, it communicates with the primary master by using the shared memory instead of sending and receiving MPI messages through the network.

## 2.4 Datasets

The main source for protein information is the Protein Data Bank (PDB) [46], where data are kept in a textual format. The PDBML format [47] has been proposed as an XML translation for PDB files. Traditional database management systems are not well suited for storing and processing biological data in this format. Several works [48, 49] extend the traditional SQL query language with a new one for allowing biologists to formulate queries. Others [50, 51] directly translate the motif identification problem to DB queries.

The solution adopted for CMS is to drop the PDB format and SQL entirely: Each protein file retrieved from the PDB is automatically transformed in a custom XML format and stored in a non-relational database [42]. The CMSXML format describes the simplified geometrical model of line segments described in Sect. 2.2. Similarities identified by a CMS kernels are stored in a similar format.

For all the experiments discussed in this paper, CMS has been executed on a selected dataset containing 1,549 CMSXML files. The dataset has been built by picking up one protein for each superfamily in PDB. In particular, we selected from the structural classification of proteins (SCOP) dataset [52] the first member in lexicographic order of each protein superfamily. A complete list of all proteins can be found in [10]. The rationale of this choice stems from the primary goal of CMS, as stated in Sect. 2, that is, the identification of previously unknown motifs in "unfamiliar" proteins.

According to Eq. 1, having a dataset with 1549 protein files, the number of all possible comparisons and thus of tasks executed by any parallel implementation is equal to $(1549 \times 1548)/2 = 1,198,926$.

### 2.5 HPC experimental results

Experiments have shown that the HPC implementation of CMS is able to scale very well when it runs on a on-premise cluster and on Marconi [10, 12], which is a powerful HPC system provided by Cineca [53]. Several profiling activities, carried out with tools such as Intel VTune, Intel Trace Analyzer and Intel Advisory, were employed to improve application performance [12]. By large the most relevant limiting factor to the performance was the high unbalance in the workload, as different protein pairs could take very different amounts of time to be processed. Indeed, as shown in [12], a CMS run on the fastest protein pair (2b1y.xml and 2hep.xml) took just $3 \times 10^{-5}$ s, while the slowest one (on 1k32.xml and 1bgl.xml) took almost $10^3$ s.[2] To make things worse, the standard deviation of execution times was equal to 2.44 s, ten times greater than the mean time.

Although it is not possible to accurately predict the completion time of a task, in [11] we showed that there is a good correlation ($\rho = 0.653$) between the size of a protein pair (as the product of the number of secondary structures in both protein files) and the final execution time. By leveraging on the above results, the initial random selection was replaced by a new workload distribution policy, named Longest Job First [12], which selects the next task to be sent for computation by predicting the task completion time. By introducing this feature, we reduced the global execution time and task idleness, and we were able to increase the Global Load Balancing factor from 0.64597 to 0.99976.

## 3 Migrating CMS to the cloud

As migrating a complex HPC application to the cloud is neither easy nor effortless, any migration attempt should start by modeling the communication pattern of the target application. Even if such model cannot be used to predict the exact communication time of the application, it can give useful hints to predict the application behavior and understand if a cloud migration is indeed worth doing.

For building a communication model, two different sets of parameters are needed: some application-related parameters, in order to characterize the communication model; and some network-related parameters, in order to characterize the interconnection network. While the application-related parameters are invariant with respect to the infrastructure on which the application is executed on, the network-related parameters are strictly related to the network infrastructure and need to be gathered on all different network layers. For this reason, a preliminary step is the identification of the target cloud architecture (Sect. 3.1).

For the purpose of this work, the number of MPI function calls and the amount of data transferred by each function were selected as application-related parameters

[2] Note that 1k32 and 1bgl share identical chains; using *a priori* biological information would greatly reduce the computational time. As stated before, however, CMS only focuses on geometrical information.

and collected using profiling tools such as Intel Trace Analyzer on the HPC implementation of CMS (Sect. 2.3). On the other hand, network-related parameters were measured used a pLogP [54] model (see Sect. 3.2) on both the HPC and the cloud environments (Sect. 3.3). Finally, the selected application-related and network-related parameters were combined together in order to actually build the communication model (Sect. 3.4).

### 3.1 Target cloud architecture

For the purpose of this work, we decided to build a cluster of Virtual Instances using Google as the cloud provider. As the economic aspect of cloud migration is not investigated in this paper, the choice of a particular provider is irrelevant. In order to provide a fair comparison, each virtual instance was configured to match as much as possible the node configuration available on Marconi, the HPC system running the HPC implementation of CMS. In particular, we built a virtual cluster with three virtual instances, each equipped with eight virtual CPUs (vCPUs). Each vCPU was mapped to a single hardware hyperthread on an Intel Xeon E5 v5 (Broadwell) running at 2.2 GHz. Each virtual instance was given 24 GB of RAM and 56 MB of L3 cache [55]. All virtual instances belonged to the same geographical zone and were interconnected together by using a private cloud network with a 2GB bandwidth guarantee for each vCPU [56, 57].

### 3.2 The plogP model

In order to build a complete communication model, a characterization of a point-to-point communication is need as a basic building brick. In the literature, several models were proposed for the purpose [58–63]. In this work, the choice fell on parameterized LogP, or pLogP [54]. This model is able to capture the relevant aspects of message passing in distributed systems, by describing a point-to-point communication with five simple parameters:

1. $P$ is the number of processors;
2. $L$ is the end-to-end latency from process to process; it includes all contributing factors such as copying data to and from network interfaces and the transfer over the physical network;
3. $o_s(m)$ is the send overhead for a message of size $m$;
4. $o_r(m)$ is the receive overhead for a message of size $m$;
5. $g(m)$ or gap is the minimum time between consecutive message transmissions or receptions; it is the reciprocal value of the end-to-end bandwidth from process to process.

A graphical representation of all pLogP parameters is shown in Fig. 4. As shown in the figure, the time spent to send a $m$-byte message can be computed as [64].

$$T = L + g(m) \tag{2}$$

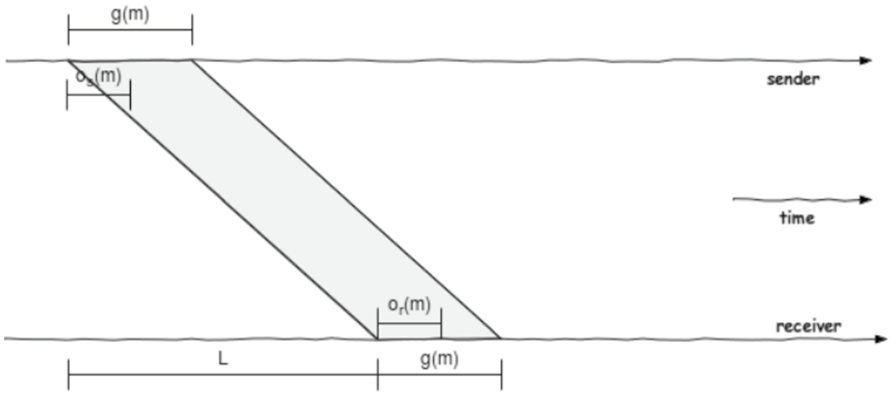


**Fig. 4** The pLogP parameters: $L$ is the end-to-end latency; $o_s(m)$ is the send overhead; $o_r(m)$ is the receive overhead; $g(m)$ is the minimum time between consecutive transmissions

**Table 1** pLogP parameters (in s) for intranode and internode communications collected on the Google Cloud Infrastructure

| | Internode | Intranode |
|---|---|---|
| $L$ | 0.0000517 | 0.0000001 |
| $g$ (4 B) | 0.0000226 | 0.0000007 |
| $g$ (6 KB) | 0.0000370 | 0.0000026 |

This formula can be used to accurately predict the time spent by a sender to send a message to a receiver. For collective operations, the formula needs to be extended [30].

## 3.3 Network characterization

Once a migration target has been defined (Sect. 3.1) and a proper model has been chosen (Sect. 3.2), it is possible to get a complete characterization of the interconnection network on the target.

Using a custom test application and the tool provided by pLogP, we discovered that: (1) the given interconnection network is symmetric and (2) the cores (i.e., the virtual CPUs) can be grouped together according to the node (i.e., the virtual instance) they belong to. According to these observations, we were able to classify all communications in two classes: *intranode* and *internode*. Intranode communication describes a communication between two processes running on two cores belonging to the same virtual instance; on the other hand, internode communication describes a communication between two processes running on two cores belonging to two different virtual instances. For a complete characterization of the interconnection network, both types of communications need to be profiled.

Table 1 shows the pLogP parameters, gathered running the tool on the cloud interconnection network. All these values have been gathered by choosing the

**Table 2** pLogP parameters (in s) for intranode and internode communications collected on the Marconi HPC system

| | Internode | Intranode |
|---|---|---|
| $L$ | 0.0000025 | 0.0000004 |
| $g$ (4 B) | 0.0000010 | 0.0000009 |
| $g$ (6 KB) | 0.0000026 | 0.0000015 |

**Table 3** Estimation of the communication time (in s) of the CMS implementation on Google Cloud, using the data of Table 1 in Eqs. 4 and 5

| | Internode | Intranode | Total |
|---|---|---|---|
| Communication number | 799,284 | 399,642 | 1,198,926 |
| Time spent to send a 4-byte message | 0.0000743 | 0.0000075 | |
| Time spent to send a 6-Kbyte message | 0.0000887 | 0.0000265 | |
| Time spent in communication | 130.283292 | 13.587828 | 143.871120 |

median computed on ten different runs. We decided to use the median instead of the mean in order to ignore any outlier. From the table, it is clear that parameters describing the internode communication are several times higher than the corresponding parameters of intranode communication.

For comparison, Table 2 shows the pLogP parameters gathered running the tool on Marconi, the HPC infrastructure running the HPC implementation of CMS. Of course, even on a real HPC system the internode communication is more expensive than the intranode one, but, as shown in the table, their relative ratio is smaller on the HPC system, and thus, internode communication has a much larger effect on the cloud implementation.

### 3.4 The communication model

Once both application and network parameters have been collected, a complete communication model can be built. Under the simplifying but yet realistic assumption that all tasks are uniformly distributed across all processes, it is possible to predict the communication time $T$ of CMS with a simple extension of Eq. 2:

$$T = T_{\text{intra}} + T_{\text{inter}} \tag{3}$$

$$T_{\text{intra}} = n_{\text{intra}} \times [L + g_{\text{intra}}(4\,\text{B}) + L + g_{\text{intra}}(6\,\text{KB})] \tag{4}$$

$$T_{\text{inter}} = n_{\text{inter}} \times [L + g_{\text{inter}}(4\,\text{B}) + L + g_{\text{inter}}(6\,\text{KB})] \tag{5}$$

where $n$ is the number of times the MPI_Send has been invoked, $L$ is the latency and $g()$ is the gap function.

Table 3 estimates both internode and intranode communication times by imputing the values of Table 1 in Eqs. 4 and 5. As shown in the table, sending a 6-Kbyte message between two processes belonging the same virtual instance is more than 33 times faster than sending the same message between two internode

**Table 4** Estimation of the communication time (in s) of the CMS implementation on Marconi, using the data of Table 2 in Eqs. 4 and 5

| | Internode | Intranode | Total |
|---|---|---|---|
| Communication number | 799,284 | 399,642 | 1,198,926 |
| Time spent to send a 4-byte message | 0.0000035 | 0.0000013 | |
| Time spent to send a 6-Kbyte message | 0.0000051 | 0.000001 | |
| Time spent in communication | 6.8738424 | 1.2388902 | 8.1127326 |

processes. The ratio becomes 99 for small messages, where latency is the dominant factor in the communication time. It becomes clear that, in order to improve the application performance, internode communication needs to be reduced. The same analysis has also been performed on Marconi and is presented in Table 4. The table shows that all parameters on Marconi are lower than the corresponding parameters on the cloud and the ratio among them is smaller.

Summarizing, sending a message having a fixed size always takes more time on the cloud than on Marconi. Furthermore, on both architectures, as expected, internode communication is more expensive than the intranode one, but the ratio between internode communication time and intranode communication time is much higher on the cloud than on Marconi. Reducing internode communications, then, is of course useful for both architecture but more beneficial for the cloud implementation, where internode communication is more expensive. However, it is critical to note that the communication time is, in both scenarios, much smaller than the total execution time (see Sect. 5).

According to the analysis just described, we can speculate that Cross Motif Search could be successfully executed on the cloud with a negligible loss of performance, especially so, if the amount of internode communication could be reduced.

## 4 Location-aware scheduling policy

As shown in the previous section, internode communication in the cloud is significantly more expensive than intranode one. In order to optimize the communication time and reduce the performance gap between cloud and HPC implementations, we have identified a new, location-aware scheduling policy for distributing tasks among the processes which will minimize internode communication and maximize intranode one.

The Longest Job First policy, defined in [12] and briefly described in Sect. 2.5, took in account only one factor: the predicted completion time of task. On the other hand, the new policy takes also into account the worker position within the virtual cluster.

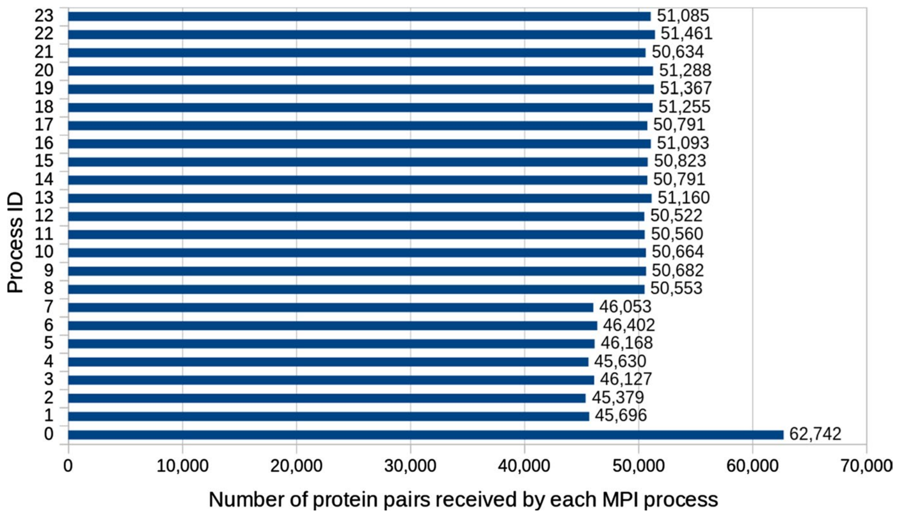


**Fig. 5** Number of messages (i.e., protein pairs) received by each worker with the old scheduling policy. Processes from 0 to 7 belong to the same virtual instance of the master process; the others belong to different virtual instances

## 4.1 Implementation

In order to implement the new policy, master and workers must establish a preliminary negotiation—an handshake—before any actual computation. In particular, each worker must send to the master its position, i.e., the identification of the virtual node on which it is running. This information is stored in a table by the master, and it is used to understand whether the worker belongs to the master instance or to a different one.

After completing the handshake phase, the computation phase starts. Exactly as in the previous implementation (Sect. 2.3), each worker sends to the master a 4-byte ready message. However, according to new policy, the master uses the data exchanged during the handshake to decide which task (i.e., which protein pair) to assign to the asking worker. If the worker belongs to the same virtual instance as the master, the protein pair having the lowest predicted computation time is sent back. Otherwise, the master sends back a protein pair having the highest predicted computation time. The rationale of the policy is to reduce as much as possible communication overhead, by placing short, frequent tasks close-by to the master.

Before introducing the new policy, the tasks were uniformly distributed across all processes. Figure 5 shows the number of tasks assigned to each process using the old, Longest Job First policy. As the scheduling algorithm is on-demand, it is impossible to predict in advance how many protein couples a process might receive and this amount changes in different runs, but it is easy to notice that the tasks were distributed quite uniformly among all processes.

According to the model presented in previous section, intranode and internode communication is directly proportional to the number of tasks, respectively, sent by

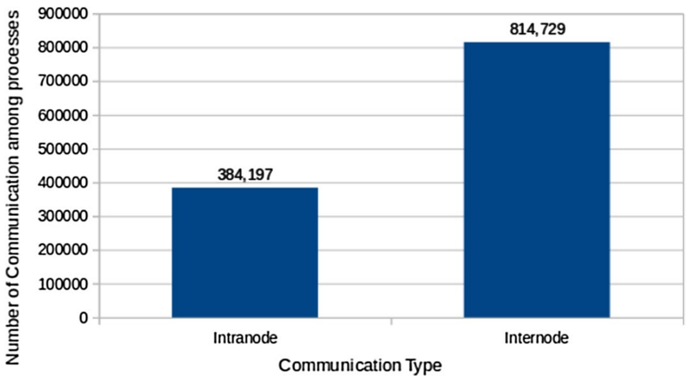


**Fig. 6** Total number of intranode and internode messages sent by the master with the old scheduling policy

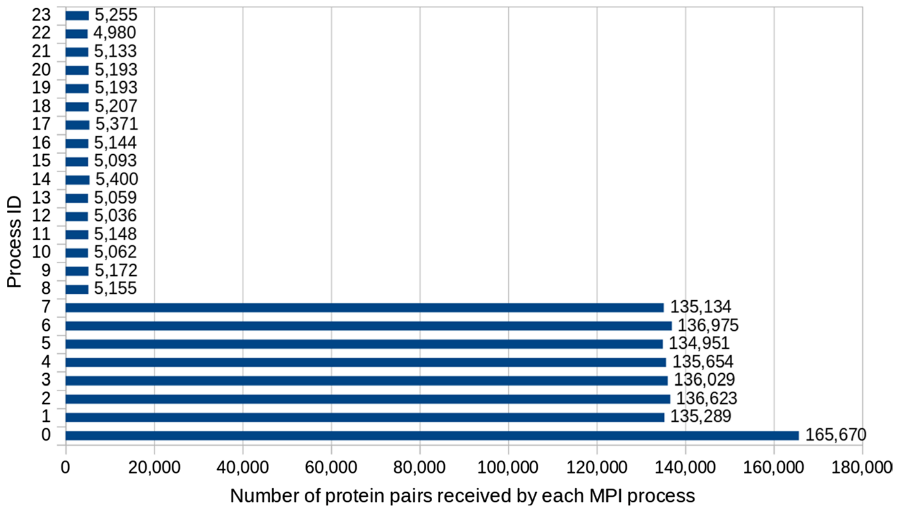


**Fig. 7** Number of messages (i.e., protein pairs) received by each worker with the new location-aware scheduling policy. Processes from 0 to 7 belong to the same virtual instance of the master process; the others belong to different virtual instances

the master to the same or to a different virtual node. For this reason, by simply summing up the amount of tasks received by each worker in Fig. 5, it is easy to compute the number of intranode and internode messages, as shown in Fig. 6.

From the figure, it is clear that the number of the internode messages is twice the number of the intranode ones. This behavior might be considered negligible for application running on a real HPC system, such as on Marconi, where the ratio between intranode and internode communication is quite low. However, for a cloud implementation, the unbalance between intranode and internode communication might compromise the application performance. Figure 7 shows how messages are

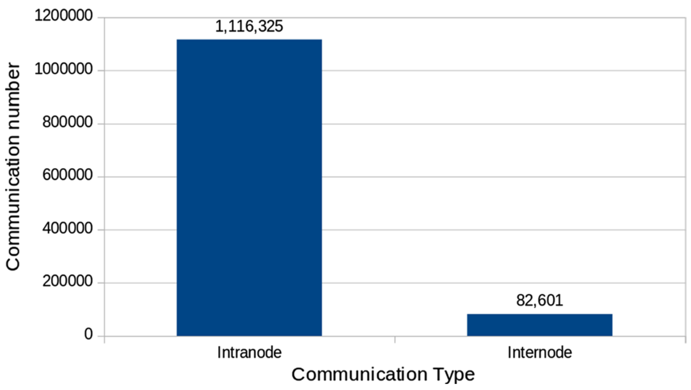


**Fig. 8** Total number of intranode and internode messages sent by the master with the new, location-aware scheduling policy

**Table 5** Communication time (in s), as predicted by the pLogP model, before and after introducing the new scheduling policy

| | Old scheduling | | New scheduling | |
|---|---|---|---|---|
| | 4 B message | 6 KB message | 4 B message | 6 KB message |
| Intranode | 0.29 | 1.01 | 0.84 | 2.96 |
| Internode | 60.53 | 72.27 | 6.14 | 7.33 |
| Total | 134.11 | | 17.26 | |

distributed to the workers according to new scheduling policy. Clearly, the internode communication has been heavily reduced, as shown in Fig. 8.

Table 5 shows the estimated values for intranode and internode communication, using the new scheduling policy. The values in the table have been obtained using the pLogP model and the results in Table 1. As shown in Table 5, the time spent in communication with the new policy is more than seven times lower than before and could greatly benefit both HPC and cloud implementation. In Sect. 5, the effects of new scheduling policy on the global load balancing will be evaluated through several experiments.

## 5 Experiments

### 5.1 Global load balancing factor

The location-aware policy introduced in Sect. 4 completely changes the workload distribution strategy, and this might bring to the conclusion that an imbalance among processes might arise. In a previous work, we evaluated the efficiency of the balancing algorithm using the Global Load Balancing Factor [11]. We showed that

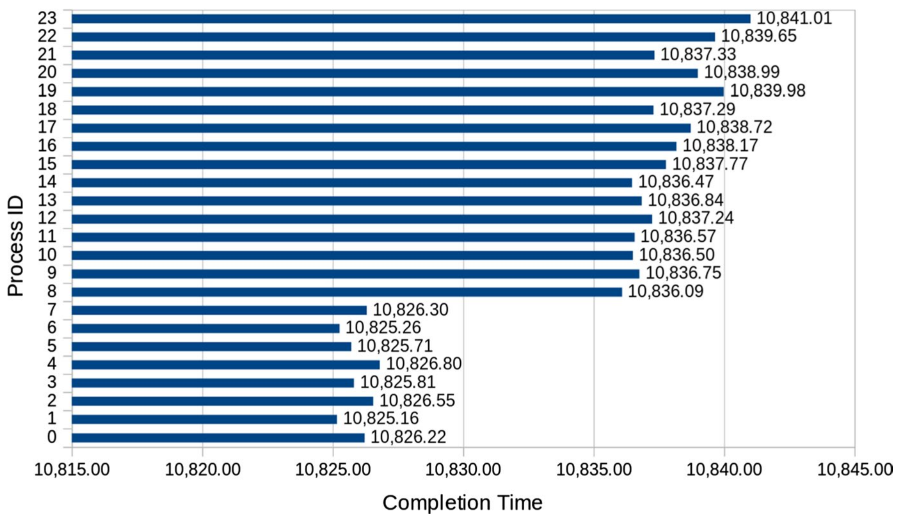


**Fig. 9** Completion time of the 24 MPI processes running on the cloud

after introducing the Longest Job First policy, the target application achieved a very good balance, with the balancing factor very close to the ideal value of 1.

To compare the effectiveness of the new scheduler, we can compute the Global Load Balancing Factor for the location-aware implementation, using a reference cloud implementation with 24 MPI processes running on the target infrastructure described in Sect. 3.1. Figure 9 shows the completion time of the 24 MPI processes. Depending on the last task received by each process, the completion time can be different. However, the figure shows that all MPI processes are well balanced even with the location-aware policy, and the Global Load Balancing Factor is still very close to the ideal value:

$$\mathrm{LB} = \mathrm{avg}(T_\mathrm{p})/\mathrm{max}(T_\mathrm{p}) = 10833.88/10841.01 = 0.99934 \tag{6}$$

where $T_\mathrm{p}$ is the time spent by a process p to compute all the received tasks.

## 5.2 Scalability and speedup

Finally, as for the HPC implementation, we studied the application scalability on the cloud infrastructure as well, increasing the number of concurrent MPI processes used to compute the dataset. As shown in Fig. 10, our application shows a good scalability. More formally, using as a baseline benchmark the performance on 16 processors, the ideal speedup for a run with 256 MPI processes will obviously be $256/16 = 16$. The actual speedup $s$ is:

$$s = T(16)/T(256) = 17,565.48/1,182.12 \approx 14.86 \tag{7}$$

where $T(16)$ and $T(256)$ are the global execution times using, respectively, 16 and 256 MPI processes.

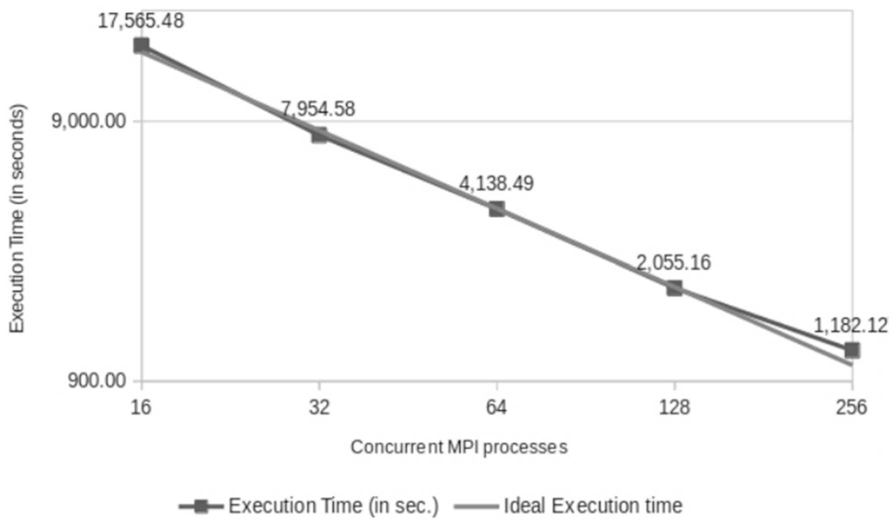


**Fig. 10** Scalability analysis for the cloud implementation, running 16 to 256 MPI processes. The measured scalability (dark gray) is compared with an ideal scenario (light gray), created by applying linear scalability to the 16-core execution baseline

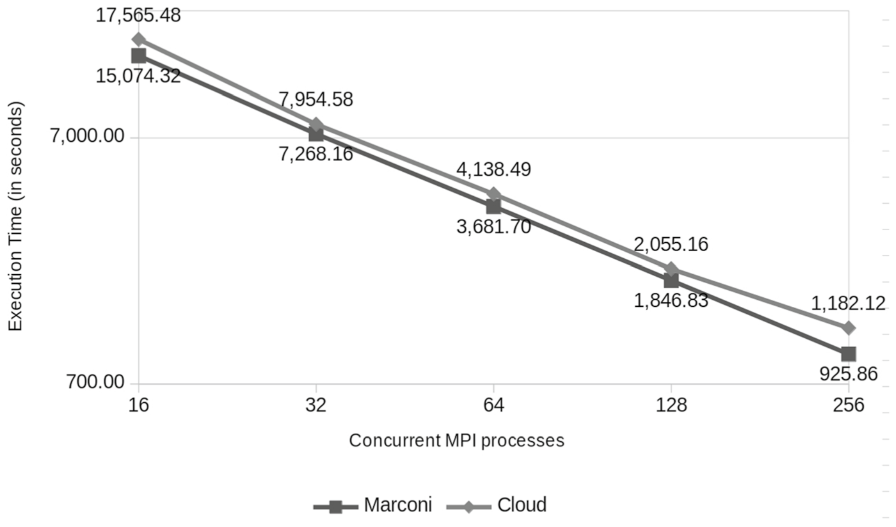


**Fig. 11** Comparing global execution time of CMS between HPC and Cloud systems

The location-aware implementation of Cross Motif Search has also been tested on Marconi. Figure 11 shows the comparison of the performance results obtained running the application on both architectures. Although the cloud infrastructure was set up to be very close to the HPC configuration, Marconi is slightly superior in raw performance. The reason is mainly due to the overhead introduced by the communication, to the virtualization layer and to the interfering jobs being run on the same physical bare metal.

## 6 Discussion and conclusions

Cross Motif Search is a computational intensive algorithm for the geometrical motif retrieval in the secondary structure of proteins, which can greatly benefit from an optimized parallel implementation.

In this paper, we have presented an in-depth analysis on the migration of the HPC implementation of Cross Motif Search to a cloud infrastructure. A complete communication model for the application has been presented, tested and compared in both the HPC and the cloud scenario. A new scheduling policy, based on the location of the task on each virtual node, has been presented and validated, in order to minimize internode communication in the cloud implementation.

The migration has been assessed through several experiments. Overall, the experiments show that the cloud implementation has a global performance profile that is not too dissimilar from the HPC implementation one. In particular, the global execution times are very close to each other using very similar node configurations and they are both able to achieve a very good scalability. Experiments also showed that the new scheduling policy is able to reduce the communication time in both implementation, without sacrificing the global load balancing.

The communication time penalty for cloud execution predicted by the model in Sect. 3.4 has been shown to be negligible, due to the low communication intensity of the CMS implementation, where message payloads are very small, infrequent, and there is no interworker communication.

The paper has surely proven that a cloud implementation for CMS is feasible. However, from the raw performance standpoint, the HPC implementation seems to be superior. Nonetheless, cloud migration could be the best choice after all for reasons beyond the performance standpoint. First of all, the assumption that the cloud vCPUs are on par with the HPC one is biased against cloud computing, because the raw computing power of a single cloud vCPU might be higher than the one of a single HPC core. Moreover, generally speaking, HPC access is time limited and may require long waiting times for each job.

A complete analysis, which is going to be our next work, should also include a complete cost comparison, which should also include an estimation of the turn-around time. Finally, we would like to note that the results presented in the paper are typical of any application which has a similar master/worker communication paradigm and thus are not limited to the specifics of the target application, but can be profitably applied to other similar cases.

## References

1. Ferretti M, Santangelo L (2018) Protein secondary structure analysis in the cloud. In: Vega-Rodrguez MA, Santander-Jimnez S, Granado-Criado JM, Badia RM (eds) Proceedings of the 6th International Workshop on Parallelism in Bioinformatics (PBio 2018). ACM, New York, pp 63–70
2. Yang H, Tate M (2012) A descriptive literature review and classification of cloud computing research. CAIS 31:2
3. Mell P, Grance T (2011) The NIST definition of cloud computing. Retrieved from http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf
4. Carlyle G, Harrell SL, Smith PM (2010) Cost-effective HPC: the community or the cloud? In: IEEE 2nd International Conference on Cloud Computing Technology and Science, Indianapolis, IN, 2010, pp 169–176
5. Hassani R, Aiatullah Md, Luksch P (2014) Improving HPC application performance in public cloud. In: IERI Procedia 10:169–176, ISSN 2212-6678
6. Mancini M, Aloisio G (2015) How advanced cloud technologies can impact and change HPC environments for simulation. In: International Conference on High Performance Computing & Simulation (HPCS), Amsterdam, 2015, pp 667–668
7. Yang T, Ma X, Mueller F (2005) Predicting parallel applications performance across platforms using partial execution. In: ACM/IEEE Supercomputing Conference
8. Chakthranont N, Khunphet P, Takano R, Ikegami T (2014) Exploring the performance impact of virtualization on an HPC cloud. In: IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, pp 426–432
9. Expsito RR, Taboada GL, Ramos S, Tourino J, Doallo R (2013) Performance analysis of HPC applications in the cloud. Fut Gen Comput Syst 29(1):218–229
10. Ferretti M, Musci M, Santangelo L (2014) A hybrid OpenMP and OpenMPI approach to geometrical motif search in proteins. In: Proceedings of the IEEE International Conference on Cluster Computing (IEEE Cluster 2014), IEEE Computer Society, 2014, pp 298–304
11. Ferretti M, Musci M, Santangelo L (2015) MPI-CMS: a hybrid parallel approach to geometrical motif search in proteins. Concurr Comput Pract Exp 27(18):5500–5516
12. Ferretti M, Santangelo L (2018) Hybrid OpenMP-MPI parallelism: porting experiments from small to large clusters. In: 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing, PDP 2018, Cambridge, UK, March 21–23, 2018. IEEE Computer Society 2018, pp 297–301
13. Ferretti M, Musci M (2013) Entire motifs search of secondary structures in proteins: a parallelization study. In: Proceedings of the 20th European MPI Users' Group Meeting. ACM
14. Drago G, Ferretti M, Musci M (2013) CCMS: A greedy approach to motif extraction. In: International Conference on Image Analysis and Processing. Springer, Berlin
15. Ferretti M, Musci M (2015) Geometrical motifs search in proteins: a parallel approach. Paral Comput 42:60–74
16. Cantoni V et al (2016) Structural motifs identification and retrieval: a geometrical approach. In: Pattern Recognition in Computational Molecular Biology: Techniques and Approaches. Wiley
17. Casavant TL, Kuhl JG (1998) A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Trans Soft Eng 14:141–154
18. Plastino A, Ribeiro CC, Rodriguez NR (2001) Load balancing algorithms for SPMD applications. Retrieved from https://pdfs.semanticscholar.org/f5d0/edd1e1e4268549e1f28f141347482ee56fea.pdf
19. Osman A, Ammar H (2002) Dynamic load balancing strategies for parallel computers. Sci Ann Cuza Univ 11:110–120
20. Amandeep K, Pawan LM (2018) A review on load balancing in cloud environment. Int J Comput Technol 17(1):7120–7125
21. Sarood O, Gupta A, Kal LV (2012) Cloud friendly load balancing for hpc applications: Preliminary work. In: 41st International Conference on Parallel Processing Workshops. IEEE
22. Rathore J, Keswani B, Rathore VS (2019) Analysis of load balancing algorithms using cloud analyst. In: Rathore V, Worring M, Mishra D, Joshi A, Maheshwari S (eds) Emerging Trends in Expert Applications and Security. Advances in Intelligent Systems and Computing, vol 841. Springer, Singapore
23. Hota A, Mohapatra S, Mohanty S (2019) Survey of different load balancing approach-based algorithms in cloud computing: a comprehensive review. In: Behera H, Nayak J, Naik B, Abraham A

(eds) Computational Intelligence in Data Mining. Advances in Intelligent Systems and Computing, vol 711. Springer, Singapore

24. Gupta A et al (2013) Improving HPC application performance in cloud through dynamic load balancing. In: 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. IEEE
25. Benchara FZ et al (2016) A new efficient distributed computing middleware based on cloud micro-services for HPC. In: 5th International Conference on Multimedia Computing and Systems (ICMCS). IEEE
26. Suh E, Narahari B, Simha R (1998) Dynamic load balancing schemes for computing accessible surface area of Protein molecules. In: Proceedings of the 5th International Conference on High Performance Computing (Cat. No. 98EX238). IEEE
27. Young WS, Brooks III CL (1995) Dynamic load balancing algorithms for replicated data molecular dynamics. J Comput Chem 16(6):715–722
28. Mrozek D, Maysiak-Mrozek B, Kapciski A (2014) Cloud4Psi: cloud computing for 3D protein structure similarity searching. Bioinformatics 30(19):2822–2825
29. Auricchio F et al (2018) Benchmarking a hemodynamics application on Intel based HPC systems. Paral Comput Everywhere 32:57
30. Ferretti M, Santangelo L (2019) Profiling hemodynamic application for parallel computing in the cloud. in: 27th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP2019)
31. Auricchio F et al (2018) Parallelizing a finite element solver in computational hemodynamics: a black box approach. Int J High Perform Comput Appl 32(3):351–362
32. Auricchio F et al (2015) Assessment of a black-box approach for a parallel finite elements solver in computational hemodynamics. In: IEEE Trustcom/BigDataSE/ISPA, vol 3. IEEE
33. Do Chuong B, Katoh K (2009) Protein multiple sequence alignment. In: Functional Proteomics. Humana Press, pp 379–413
34. Holm L, Sander C (1993) Protein structure comparison by alignment of distance matrices. J Mol Biol 233(1):123–138
35. Shi S et al (2007) Searching for three-dimensional secondary structural patterns in proteins with ProSMoS. Bioinformatics 23(11):1331–1338
36. Shi S, Chitturi B, Grishin NV (2009) ProSMoS server: a pattern-based search using interaction matrix representation of protein structures. Nucl Acids Res 37(suppl2):W526–W531
37. Hutchinson EG, Thornton Janet M (1996) PROMOTIF—a program to identify and analyze structural motifs in proteins. Prot Sci 5(2):212–220
38. Dror O et al (2003) MASS: multiple structural alignment by secondary structures. Bioinformatics 19(suppl1):i95–i104
39. Krissinel E, Henrick K (2004) Secondary-structure matching (SSM), a new tool for fast protein structure alignment in three dimensions. Acta Crystallogr Sect D 60(12):2256–2268
40. Aung Z, Li J (2007) Mining super-secondary structure motifs from 3d protein structures: a sequence order independent approach. Genome Inform 19:1526
41. Cantoni V et al (2014) Protein motif retrieval by secondary structure element geometry and biological features saliency. In: 25th International Workshop on Database and Expert Systems Applications. IEEE
42. Argentieri T, Cantoni V, Musci M (2017) Extending cross motif search with heuristic data mining. In: 28th International Workshop on Database and Expert Systems Applications (DEXA). IEEE
43. Musci M, Ferretti M (2018) Mining geometrical motifs co-occurrences in the CMS dataset. In: International Conference on Database and Expert Systems Applications. Springer, Cham
44. Ballard DH (1981) Generalizing the Hough transform to detect arbitrary shapes. Pattern Recognit 13(2):111–122, ISSN 0031-3203,
45. Argentieri T, Cantoni V, Musci M (2016) MotifVisualizer: an interdisciplinary GUI for geometrical motif retrieval in proteins. In: 27th International Workshop on Database and Expert Systems Applications (DEXA). IEEE
46. Protein Data Bank. 2019, March 6. Retrieved from https://www.rcsb.org
47. Wesbrook J, Ito N, Nakamura H, Henrick K, Berman HM (2004) PDBML: the representation of archival macromolecular structure data in XML. Bioinformatics 21(7):988–992
48. Tata S, Friedman JS, Swaroop A (2006) Declarative querying for biological sequences. In: 22nd International Conference on Data Engineering (ICDE'06). IEEE

49. Mrozek D et al (2016) An efficient and flexible scanning of databases of protein secondary structures. J Intell Inform Syst 46(1):213–233
50. Hammel L, Patel JM (2002) Searching on the secondary structure of protein sequences. In: VLDB'02: Proceedings of the 28th International Conference on Very Large Databases. Morgan Kaufmann
51. Wang Y, Sunderraman Rr, Tian H (2006) A domain specific data management architecture for protein structure data. In: International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE
52. Murzin Alexey G et al (1995) SCOP: a structural classification of proteins database for the investigation of sequences and structures. J Mol Biol 247(4):536–540
53. Marconi (2017) the new Tier-0 system. 2017, July 21. Retrieved from http://hpc.cineca.it/hardware/marconi
54. Kielmann T, Bal H E, Verstoep K (2000) Fast measurement of LogP parameters for message passing platforms. In: International Parallel and Distributed Processing Symposium. Springer, Berlin
55. Machined types. 2018, May 16. Retrieved from https://cloud.google.com/compute/docs/machine-types
56. Advanced VPC Concept. 2018, December 17. Retrieved from https://cloud.google.com/vpc/docs/advanced-vpc
57. Quota. 2019, March 06. Retrieved from https://cloud.google.com/vpc/docs/quota
58. Nomura A, Matsuba H, Ishikawa Y (2007) Network performance model for TCP/IP based cluster computing. In: IEEE International Conference on Cluster Computing, Austin, TX, 2007, pp 194–203
59. Li L, Zhang X, Feng J, Dong X (2010) mPlogP: a parallel computation model for heterogeneous multi-core computer. In: 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, Melbourne, VIC, 2010, pp 679–684
60. Hoefler T, Mehlan T, Lumsdaine A, Rehm W (2007) Netgauge: a network performance measurement framework. In: Perrott R, Chapman BM, Subhlok J, de Mello RF, Yang LT (eds) High Performance Computing and Communications. HPCC 2007. Lecture Notes in Computer Science, vol 4782. Springer, Berlin
61. Hockney R (1994) The communication challenge for MPP: Intel Paragon and Meiko CS-2. Parallel Comput 20(3):389–398
62. Alexandrov A, Ionescu MF, Schauser KE, Scheiman C (1995) LogGP: incorporating long messages into the LogP model. In: Proceedings of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures. ACM Press, New York, pp 95–105
63. Culler D, Karp R, Patterson D, Sahay A, Schauser KE, Santos E, Subramonian R, von Eicken T (1993) LogP: towards a realistic model of parallel computation. In: Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. ACM Press, New York, p 112
64. Steffenel LA, Mounie G (2008) A framework for adaptive collective communications for heterogeneous hierarchical computing systems. J Comput Syst Sci 74(6):1082–1093