Josep, M. [et al.]. GOPHER, an HPC framework for large scale graph exploration and inference. A: International Conference on High Performance Computing. "High Performance Computing: ISC High Performance 2020 International Workshops: Frankfurt, Germany, June 21–25, 2020: revised selected papers". Berlín: Springer, 2020, p. 211-222. ISBN 978-3-030-59851-8.
The final authenticated version is available online at https://doi.org/10.1007/978-3-030-59851-8_13

GOPHER, an HPC framework for large scale graph exploration and inference

Marc Josep-Fabregó[1], Xavier Teruel[1], Victor Gimenez-Abalos[1], Davide Cirillo[1], Dario Garcia-Gasulla[1], Sergio Alvarez-Napagao[1], Marta García-Gasulla[1], Eduard Ayguadé[1], and Alfonso Valencia[1,2]

[1] Barcelona Supercomputing Center (BSC)
C/ Jordi Girona 29, 08034, Barcelona, Spain
marc.josep, xavier.teruel, victor.gimenez, davide.cirillo, dario.garcia, sergio.alvarez, marta.garcia, eduard.ayguade, alfonso.valencia@bsc.es
[2] ICREA
Pg. Lluís Companys 23, 08010, Barcelona, Spain

Abstract. Biological ontologies, such as the *Human Phenotype Ontology* (HPO) and the *Gene Ontology* (GO), are extensively used in biomedical research to investigate the complex relationship that exists between the phenome and the genome. The interpretation of the encoded information requires methods that efficiently interoperate between multiple ontologies providing molecular details of disease-related features. To this aim, we present *GenOtype PHenotype ExplOrer* (GOPHER), a framework to infer associations between HPO and GO terms harnessing machine learning and large-scale parallelism and scalability in High-Performance Computing. The method enables to map genotypic features to phenotypic features thus providing a valid tool for bridging functional and pathological annotations. GOPHER can improve the interpretation of molecular processes involved in pathological conditions, displaying a vast range of applications in biomedicine.

Keywords: Biological ontologies, Genomics, ML, HPC, Graph exploration.

1 Introduction

Understanding the complex processes taking place in a cell or disease requires powerful computational frameworks, able to effectively provide meaningful interpretations of large volumes of high-throughput data and clinical information [6]. In the grand challenge of biomedical data integration and interpretation, biological ontologies are recognized as essential tools [11]. An ontology is a domain-specific knowledge formalization, based on sets of entities and relations [16]. Two of the most popular biological ontologies are the Human Phenotype Ontology (HPO) [12], describing phenome abnormalities, and the Gene Ontology (GO) [20], describing genome activities.

The integration of multiple ontologies, *i.e.*, ontology mapping or alignment, is posing great challenges to Artificial Intelligence (AI) [5]. Despite substantial

efforts have been put on the integration of ontology-based biological information [17], computationally tractable approaches exploiting the interconnectivity between multiple large-scale ontological graphs still needs substantial investigation. To this aim, it is crucial to design HPC solutions and novel parallel algorithms that support fine-grained parallelism, while overcoming memory costs.

We develop and evaluate GOPHER, a system for the efficient traversing and exhaustive path enumeration in interconnected biological ontologies, enabled by the large-scale parallelism and scalability of HPC. In addition to efficient graph exploration functionalities, GOPHER harnesses machine learning to infer a precise mapping between given ontologies, allowing knowledge processing and discovery beyond limited cross-references. We applied GOPHER to study associations between disease-related phenotypic features and distinct molecular processes in humans, as well as in other model organisms (e.g., mouse and fruit fly). The approach exploits a very simple and yet very strong principle of biological ontologies: Preferential attachment, also known as *rich get richer*. The huge accuracy obtained in our experiments illustrates the utility of such property.

2 Related Work and Context

Biological ontologies are widely used. For instance, the functional interpretation of sets of genes is commonly achieved throught statistical enrichments of ontological annotations, such as GO terms [15]. Given the pervasive application of biological ontologies in the biomedical area, community efforts like the Open Biomedical and Biological Ontologies (OBO) Foundry [18] have been created to disseminate best practices and curated corpora of ontologies.

A pivotal application of biological ontologies is to study the relationships between phenotypic and genotypic characteristics [22,9]. While the genome refers to the full set of genetic material, the phenome is defined as the totality of all traits expressed by an organism [1]. Finding associations between the phenome and genome is of utmost priority in biomedicine as it could lead to the identification of the molecular drivers underlying human diseases.

Although genome-wide association studies (GWAS) have been carried out to dissect phenome-genome associations [19], biological complexity [3] and lack of consensus on pathogenicity and susceptibility [4] entail great limitations. By querying on annotated datasets to infer novel associations, the grand challenge of characterizing phenome-genome relationships would greatly benefit from the fine mapping of ontological terms that tools such as GOPHER are able to generate.

Along the years, *Artificial Intelligence* (AI) approaches have enabled to model the behavior of a given system or agent. Recently, data oriented modeling solutions have been gaining propularity and skill, thanks to the increasing digitalization of data. Nowadays, its increasingly frequent to see how *simple models and a lot of data trump more elaborate models based on less data* [10]. Furthermore, when working with biological data, complex and detailed modeling solutions can be plain unfeasible. For this reason, in this work we chose to produce a very simple modeling solution which is based on the most basic aspects of inter-

action, empowered by a network structure thoroughly refined by experts along years. This is made possible by the power, scalability and parallelism provided by *High-Performance Computing* (HPC). A series of popular graph processing frameworks already exist (*e.g.*, igraph, networkX), however, since our task is highly specific, we choose to implement an ad-hoc solution which can be thoroughly optimized.

HPC systems allow to process data and run complex computations at a high degree of productivity (in terms of runtime speed, memory usage, or allowed storage). These HPC solutions are commonly based on supercomputers, containing thousands of compute nodes (e.g., processors) working together to complete one single task (i.e., parallel processing).

In order to manage such a complex scenario exist a vast number of tools that ease the use of HPC systems by abstracting the user from the underlying architecture: the *Message Passing Interface (MPI)* [13] is the most used parallel programming model in distributed systems (where memory is not shared, and thus the data must be explicitly sent by messages); the *Open Multi-Processing (OpenMP)* [14] is the standard de facto parallel programming model for shared memory environments (where the communication is done implicitly via the same Memory Address Space); and/or the *OpenMP SuperScalar (OmpSs)* [7], a task-based parallel programming model, considered as a forerunner for OpenMP (its ideas have been transferred into the OpenMP standard on several occasions).

3 GOPHER: Analysis, Design, and Implementation

Our goal is to predict when a pair of ontology terms, henceforth referred to as <genotype, phenotype>, could be directly associated through an intermediate gene. We build a model to represent direct association, and another to represent the lack of direct association. Given both models we can measure the probability with which a new pair belongs to either one of those. To build these models we use examples from the ontologies. Models are designed to strive for simplicity, prioritizing volume over complexity. To that end, we estimate the distribution of distinct paths (of a given maximum length) that connect <genotype, phenotype> pairs for both sets. We hypothesize that the type of path (the ordered sequence of vertex type traversed) may contribute to an accurate modeling.

For simplicity, we assume that the frequency of type of path follows a Gaussian distribution, which has a *mean* and *standard deviation* that can be estimated empirically. We note that this assumption is taken due to the unimodal distribution empirically observed, and we use the pervasive Gaussian to be able to perform inference. Therefore, given a <type of path, number of paths>, we can compute the probability of that number of paths being generated by each of the two corresponding gaussians (one for each of the two sets) using Eq. 1.

$$P(c|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{1}$$

By making the quotient of these two probabilities we obtain the *odds* of each path type. We model the odds of connection of a new pair as the product of

each path-specific odds, for a finite number of path types (*i.e.*, all paths up to a maximum length) as seen in Eq. 2.

$$Odds(connected|Counts) = \prod_{pt \in pathtypes} \frac{P(Count(pt)|\mu_{con}(pt), \sigma_{con}(pt))}{P(Count(pt)|\mu_{discon}(pt), \sigma_{discon}(pt))} \tag{2}$$

3.1 Analysis of requirements

Modelling the association between <genotype,phenotype> pairs is subject to scientific investigation. It is desirable that both the design and the implementation of the system is highly programmable. Besides programmability, we also pursue a modular design to enable possible extensions of the system components.

GOPHER offers an efficient solution to modelling an embarrasingly parallel problem. Indeed, building the model is achieved by sampling pairs of directly associated (or not) pairs and finding all possible paths of a maximum length taking from one element of the pair to the other. Meanwhile, the total number of possible pairs to sample from is huge, which calls for a parallel approach.

3.2 Design and implementation

Graph topology We create a comprehensive graph based on the relationships between phenotypes, genotypes, and genes. Figure 1 shows an example of this graph's structure. Ontology relations between phenotypes (blue nodes) and genotypes (green nodes) are shown as pointed arrows, and association to genes (red nodes) as dashed lines.

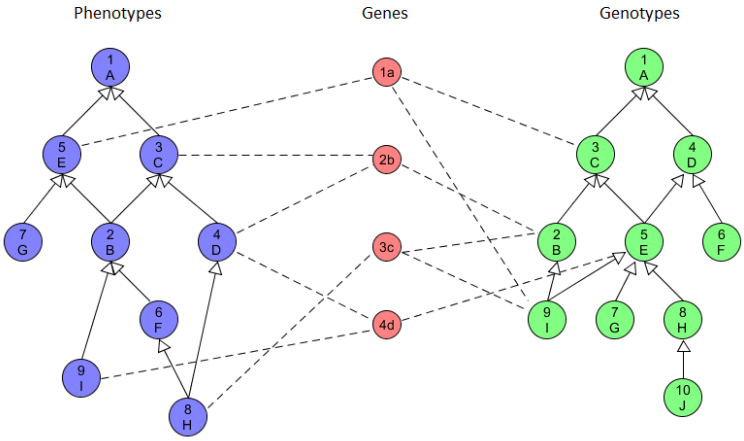
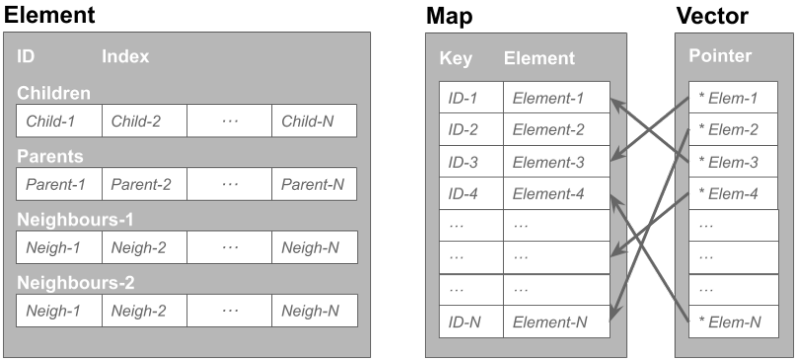


Fig. 1: Example of the graph's structure.

We define a *type of path* as the sequence of steps between two nodes in the graph, according to the nature of each one. We label phenotypes as 'p', genotypes as 'g' and genes as 'G'. Then, a path of type *ppGg* starts on a phenotype (**p**), connects to another phenotype (**p**) (either its parent or its child), which is associated to a gene (**G**), which in its turn is finally associated to a genotype (**g**). We also define as a "directly associated pair" a pair of phenotype and genotype where there exists at least one path of type *pGg*.

Data structures An element is the basic component of the graph, and it can either be a phenotype, a genotype or a gene. In terms of implementation and of element structure, there is no difference between phenotypes, genotypes, and genes. As depicted in Figure 2a, an element has the following fields:



(a) Structure of an element.

(b) Map and vector structures.

Fig. 2: Ontology and pool structures used to represent a GOPHER graph.

- **Id:** The identifier is an integer defined by the input files. Not all sequential integer values need to be defined (i.e., phenotype 4 may exist without the need of the existence of phenotype 3).
- **Index:** The index is an integer defined by the application. The index can take values from zero up to the total number of nodes minus one (no gaps, no duplications within each ontology).
- **Children:** Vector of pointers to all the children of the element. If the element is a leaf or a gene, this vector will be empty.
- **Parents:** Vector of pointers to all the parents of the element. If the element is a root or a gene, this vector will be empty.
- **Neighbours:** According to the element type, these vectors can have two differents uses:
  - If the element is a phenotype or genotype, it will have a single vector containing pointers to its associated genes. It may be empty if the element has no associations.

- If the element is a gene, it will have two vectors, one containing pointers to its phenotypes associations, and another one pointing the its genotypes associations. Vectors may be empty, if the gene has no associations to one of the two ontologies.

Each ontology and the gene pool are defined by two different types of structures (see Figure 2b). First, the **Map** structure allows to access any element of the graph providing its *Id* as the key value. Second, the **Vector** structure allows to access any element of the graph providing its *Index* as a key value (i.e., which actually matches with its position in the vector)[3].

Algorithms GOPHER's core algorithm is a recursive exploring function, which searches for all existing paths between a given pair of elements. Using this core algorithm, we centered on two different functionalities:

- **Number of paths for each path type:** for each phenotype-genotype pair, we search for the number of existing paths following each of the possible path type patterns up to a given maximum length (e.g., 5).
- **Average number of paths of a given path type:** for each phenotype-genotype pair, we search for the number of existing paths following the pattern of a given path type. The obtained data is then aggregated in order to produce the average and the standard deviation.

Due to the high computational cost of these functions, we implemented the option to only explore a random part of those pairs.

MPI Parallelization To avoid unneeded MPI communication between processes, there is no graph data distribution; all processes read the input files an populate a whole graph each. Our first MPI parallelization approach consists of equally distribute one of the two ontologies (e.g., phenotypes) between the processes, and each process explore the pairs starting from its assigned elements.

When finding the number of paths for each path type, no additional communication is needed (see Algorithm 1). Instead, when we want the average number of paths of a certain path's type and its standard deviation, we need to share the results between processes to calculate the average values after all processes have finished its assigned iterations. Algorithm 2 describes this functionality. We can see that there are two communication phases. The first one adds up the obtained number of paths and pairs from all processes, so each process can calculate the global average number of paths per pair. When all processes have the average value, each one calculates its local standard deviation. During the second communication phase, the local standard deviations values are reduced at the first process, which afterward calculates the global standard deviation.

This first approach of distributing phenotypes among MPI processes produces a huge load imbalance between processes. We tracked the origin from this

[3] Vectors are parallel-friendly structures that allow to easily split the elements among different compute elements.

Algorithm 1 MPI parallelization without communication

```
1: chSize ← Ontology 1 size / num Ranks
2: (start, end) ← (my Rank ∗ chSize, my Rank ∗ chSize + chSize)
3: for i ← start, end do
4:     for j ← 0, Ontology 2 size do
5:         for k ← 0, num path types do
6:             Search Paths(Ontology 1(i), Ontology 2(j), path type(k))
```

Algorithm 2 MPI parallelization with communication

```
1: chSize ← Ontology 1 size / num Ranks
2: (start, end) ← (my Rank ∗ chSize, my Rank ∗ chSize + chSize)
3: (nPaths, nPairs) ← (0, 0)
4: for i ← start, end do
5:     for j ← 0, Ontology 2 size do
6:         nPaths ← nPaths+ Search Paths(Ontology 1(i), Ontology 2(j), path type)
7:         nPairs ← nPairs + 1
8: MPI_AllReduce (nPaths, nPairs)
9: (average, st dev) ← (nPaths/nPairs, Calculate Local St Dev)
10: MPI_Reduce (st dev)
11: if my Rank = 0 then
12:     st dev ← Calculate Overall St Dev
```

imbalance down to the number of direct connections per element. As can be seen in Figure 3, there is a correlation between the average number of first-degree connections the elements assigned to a processor have and the time it takes to explore the paths starting from these elements.

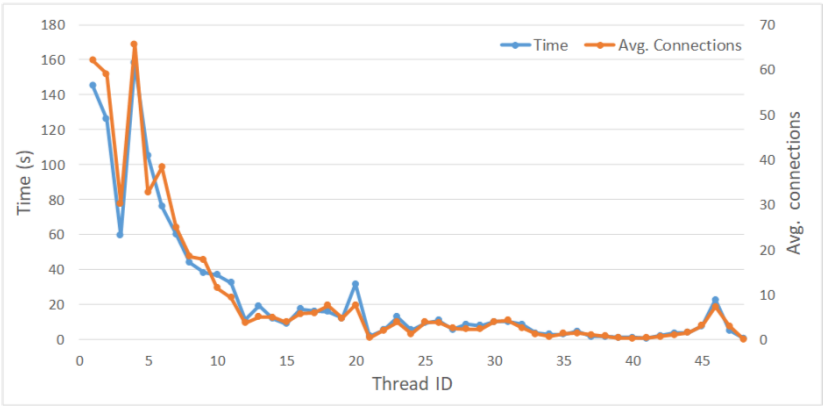


Fig. 3: Correlation between the average number of connections per element and the useful execution time per thread ID.

To improve the load balance, we change the work distribution policy. As described in Algorithm 3, the new approach distributes the elements from the starting ontology among processes based on the number of the first-degree connections. First, we calculate the total number of connections, and then we distribute the phenotypes between the MPI processes trying to keep this number balanced among processes. In Section 4 we will compare these two versions.

Algorithm 3 MPI parallelization with new work distribution

```
1: total work ← 0
2: for i ← 0, Ontology 1 size do
3:     total work ← total work + connections(i)
4: chSize ← total work / num Ranks
5: (start, end) ← (begin(my Rank, chSize), end(my Rank, chSize))
```

OpenMP/OmpSs Parallelization A second level of parallelism based in shared memory has been implemented to palliate even more the imbalance problem previously described. OpenMP and other parallel programming models based on shared memory are inherently easier to load balance, as the threads share memory is easy to redistribute work among them without data exchange.

In our case we use OmpSs, as it offers the same benefits as OpenMP but it simplifies the use of task reductions and, in addition, it has better interoperability with respect to future techniques we also want to analyze in the future (see Section 5). Our approach is to create a task per *path exploration* function call.

Algorithm 4 OmpSs parallelization with reduction

```
1: (nPaths, nPairs) ← (0, 0)
2: for i ← start, end do
3:     for j ← 0, Ontology 2 size do
4:         # pragma omp task reduction(+, nPaths)
5:         nPaths ← nPaths+ Search Paths(Ontology 1(i), Ontology 2(j), path type)
6:         nPairs ← nPairs + 1
7: # pragma omp taskwait / barrier
8: MPI All Reduce (num paths, num pairs)
9: (average, st dev) ← (nPaths/nPairs, 0)
10: # pragma omp parallel for reduction(+, st dev)
11: for i ← 0, start − end do
12:     st dev ← st dev + (nPaths(i) − average)^2
13: MPI Reduce (st dev)
```

In addition, when we calculate the average number of paths of a certain type, we need to perform a local reduction on the number of found paths. In this case, we also parallelize the computation of the standard deviation. This approach can be seen in Algorithm 4.

4 Experimental Results

We obtained all the results on the *MareNostrum IV* system located at the Barcelona Supercomputing Center. Each node contains two Intel Xeon Platinum 8160, each one with 24 processors running at 2.1 GHz and 33 MB L3 Cache. Memory is organized in two NUMA sockets (i.e., one socket per processor), with a total amount of 192GB per socket (*high-mem* nodes).

For software, we used: *GNU Compilers Collection* (gcc) version 7.2.0, *Mercurium* source-to-source compiler (mcxx) version 2.3.0, *Nanos++* Runtime Library version 0.16a, and the OpenMPI Message Passage Interface version 3.1.1.

4.1 Evaluation of model results

As introduced in Section 3.2, we estimate the parametres of the gaussians for each path type, so that we can perform inference on unseen pairs by using the odds ratio. To validate we use the *Receiver Operating Characteristic* (ROC) and *Precision-Recall* (PR) curves, which illustrate overall performance. These curves are typically evaluated through the *area-under-the-curve* (AUC). To validate the outcomes of GOPHER, we evaluate under the following conditions:

- **Ontologies:** GO and HPO, only human, version: January 2019.
- **Path size:** all types of paths with size 4 or 5 elements.
- **Pairs nature:** from phenotypes to genotypes.
- **Samples:** 85,750 randomly sampled pairs.
- **Direct edge removal:** yes[4].

We estimate the components of the gaussians required for classification using an equal number of connected and disconnected pairs. To validate the classification results we extract the ROC and PR curves from the odds ratio obtained from Eq. 2, from 85,750 pairs of each type (different from the previous ones). The AUC of both curves is 0.96, significantly close to perfect performance (*i.e.*, 1). This means that, even though we removed relevant information (the edges connecting pairs directly in the graph), our classifier is able to correctly discriminate connected from disconnected pairs with minimal error. Further experimentation on older versions of the ontology is not conducted, as these are known to include a significant amount of noise [21]

4.2 Performance results

Performance experiments have been executed with the algorithm *Number of paths for each path type* (as described in Section 3.2). The decision is based upon the complexity of the function, which clearly shows the imbalance problem between MPI processes. It is also the most relevant function for our studies. For experimenting purposes, we have run this function with the following parameters:

[4] Path frequencies for connected pairs are computed without considering the edges that directly connect the phenotype and genotype to the same gene, to avoid biasing the model towards already existing pairs.

- **Ontologies:** GO and HPO, only human, version: January 2019.
- **Path size:** all types of paths up to a size of 5 elements.
- **Pairs nature:** from phenotypes to genotypes.
- **Samples:** 100,000 randomly sampled pairs (constant seed).
- **Direct edge removal:** yes.

The timing results have been collected only from the computation (graph exploration) phase, since it takes most of the execution time in relevant cases. Previous phases, such as graph population and work distribution between processes, have been ignored.

Both versions of the algorithm have been tested and compared. In the initial version (labeled as *baseline* in Figure 4), we equally distribute elements from the origin ontology just considering the number of elements (see Algorihthms 1 and 2). While in the second version, labeled as *balancing* in the same Figure, we try to equally distribute the number of direct connection from the origin ontology (see Algorithm 3) rather than just the number of elements.

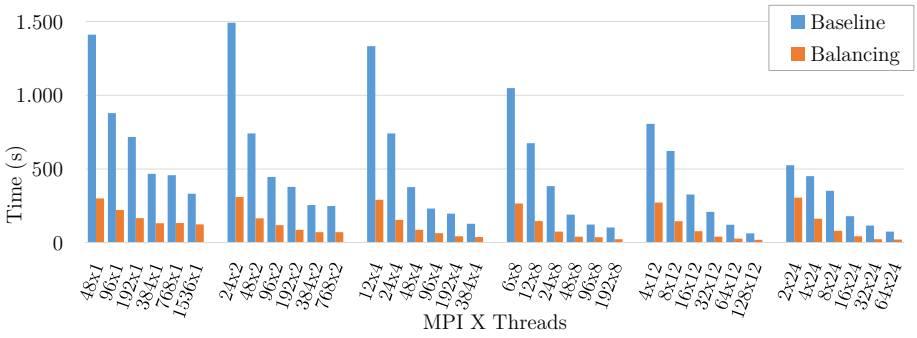


Fig. 4: Overall performance results (including baseline and balancing).

When we execute enabling none of those optimizations, shown as blue bars in Figure 4, in all the case we achieve our better results when we running a MPI-Threads configuration which uses a high number of threads (and, therefore, a small number of MPI processes). This is because imbalance problems in GOPHER reside mostly in its MPI parallelization, not in OmpSs'.

When we apply the MPI balancing technique, depicted in orange bars in Figure 4, we achieve an average speedup of 4,24x (comparing *balancing* with *non-balancing* versions) for the cases with 1, 2 and 4 threads for each MPI processes, and of 3,21x for the case with 24 threads per MPI process (mostly due to the lower baseline). When executing within a single node (i.e., first pair of columns in each group: 48x1, 24x2, etc.) is when it yields a higher impact, showing no significant differences among the different MPI-Threads configurations. As we increase the number of nodes, this version produces better results with higher thread counts, the gap getting wider with each additional node.

5 Conclusions and Future Work

In this paper we introduce the GOPHER framework for large graph exploration and inference, specially designed to run on HPC systems. GOPHER is developed to investigate the relationship between the phenome and genome using machine learning techniques to infer these complex relationships. In particular, it enables to estimate the likelihood that two ontology terms are associated when missing a direct connection through a co-annotated gene.

The work presented is extremely interdisciplinary, starting at understanding the biological questions that we want to answer through *preferential attachment*. We use a machine learning approach to infer associations between HPO and GO terms while working on large graphs. Built on top of an HPC oriented framework, designed to be modular and adaptable to solve a broad range of questions regarding a variety of ontologies.

We show that our approach obtains an AUC score of 0.96 over 1. We have also studied the parallel performance of GOPHER detecting that the main issue is related to the inherent load imbalance produced by the disparity in the number of connections. To address this issue we present an improved load balancing implementation, the evaluation shows that the load balancing implementation can overcome the performance loss due to the different number of connections and that it can scale up to 32 nodes with a relative speed-up of 4.24x.

This work opens a wide range of future work opportunities, we plan to study in detail the performance of GOPHER to find optimization opportunities in different architectures, including the use of a *Dynamic Load Balancing* library [2,8]. We will apply GOPHER to actionable use cases, such as anticancer treatment recommendations, as well as other biological ontologies, such as those of key model organisms (mouse and fruitfly).

Acknowledgements

This work has been developed with the support of the Severo Ochoa Program (SEV-2015-0493); the Spanish Ministry of Science and Innovation (TIN2015-65316-P); and the Joint Study Agreement no. W156463 under the IBM/BSC Deep Learning Center agreement.

References

1. What exactly are genomes, genotypes and phenotypes? and what about phenomes? Journal of Theoretical Biology **186**(1), 55–63 (1997)
2. Hints to improve automatic load balancing with lewi for hybrid applications. Journal of Parallel and Distributed Computing **74**(9), 2781–2794 (2014)
3. Embracing complex associations in common traits: Critical considerations for precision medicine. Trends in Genetics **32**(8), 470–484 (2016)
4. Evaluating the clinical validity of gene-disease associations: An evidence-based framework developed by the clinical genome resource. The American Journal of Human Genetics **100**(6), 895 – 906 (2017)

5. Choi, N., Song, I.Y., Han, H.: A survey on ontology mapping. ACM Sigmod Record **35**(3), 34–41 (2006)
6. Cirillo, D., Valencia, A.: Big data analytics for personalized medicine. Current Opinion in Biotechnology **58**, 161–167 (2019). https://doi.org/10.1016/j.copbio.2019.03.004, SBN
7. Duran, A., Ayguadé, E., Badia, R.M., et al.: OmpSs: A proposal for programming heterogeneous multi-core architectures. Parallel Processing Letters **21**(2), 173–193 (2011)
8. Garcia-Gasulla, M., Josep-Fabrego, M., Eguzkitza, B., Mantovani, F.: Computational fluid and particle dynamics simulations for respiratory system: Runtime optimization on an arm cluster. In: Proceedings of the 47th International Conference on Parallel Processing Companion. p. 11. ACM (2018)
9. Gkoutos, G.V., Schofield, P.N., Hoehndorf, R.: The anatomy of phenotype ontologies: principles, properties and applications. Briefings in Bioinformatics **19**(5), 1008–1021 (04 2017)
10. Halevy, A., Norvig, P., Pereira, F.: The unreasonable effectiveness of data. IEEE Intelligent Systems **24**(2), 8–12 (2009)
11. Hoehndorf, R., Schofield, P.N., Gkoutos, G.V.: The role of ontologies in biological and biomedical research: a functional perspective. Briefings in Bioinformatics **16**(6), 1069–1080 (04 2015). https://doi.org/10.1093/bib/bbv011
12. Köhler, S., Carmody, L., Vasilevsky, N., et al.: Expansion of the Human Phenotype Ontology (HPO) knowledge base and resources. Nucleic Acids Research **47**(D1), D1018–D1027 (11 2018). https://doi.org/10.1093/nar/gky1105
13. Message Passing Interface Forum: MPI: A message-passing interface standard. Version 3.1. University of Tennessee (Jun 2015)
14. OpenMP Architecture Review Board: OpenMP Application Programming Interface, version 5.0 (2018)
15. Rhee, S.Y., Wood, V., Dolinski, K., Draghici, S.: Use and misuse of the gene ontology annotations. Nature Reviews Genetics **9**, 509–515 (2008)
16. Schulze-Kremer, S.: Ontologies for molecular biology and bioinformatics. In silico biology **2**, 179–93 (02 2002)
17. Shefchek, K.A., Harris, N.L., Gargano, M., et al.: The Monarch Initiative in 2019: an integrative data and analytic platform connecting phenotypes to genotypes across species. Nucleic Acids Research **48**(D1), D704–D715 (11 2019). https://doi.org/10.1093/nar/gkz997
18. Smith, B., Ashburner, M., Rosse, C., Bard, J., et al.: The obo foundry: Coordinated evolution of ontologies to support biomedical data integration. Nature biotechnology **25**, 1251–5 (12 2007)
19. Tam, V., Patel, N., Turcotte, M., Bossé, Y., Paré, G., Meyre, D.: Benefits and limitations of genome-wide association studies. Nature Reviews Genetics (05 2019)
20. The Gene Ontology Consortium: The Gene Ontology Resource: 20 years and still GOing strong. Nucleic Acids Research **47**(D1), D330–D338 (11 2018)
21. Tomczak, A., Mortensen, J.M., Winnenburg, R., Liu, C., Alessi, D.T., Swamy, V., Vallania, F., Lofgren, S., Haynes, W., Shah, N.H., et al.: Interpretation of biological experiments changes with evolution of the gene ontology and its annotations. Scientific reports **8**(1), 1–10 (2018)
22. Zhang, W., Zhang, H., Yang, H., et al.: Computational resources associating diseases with genotypes, phenotypes and exposures. Briefings in Bioinformatics **20**(6), 2098–2115 (08 2018)