Volunteer Computing Approach for the Collaborative Simulation of Electrophysiological Models

Joel Castro*, Violeta Monasterio* and Jesús Carro*†‡
*School of Architecture and Technology, Universidad San Jorge, Zaragoza, SPAIN
†Aragon Institute for Engineering Research (I3A), Universidad de Zaragoza, Zaragoza, SPAIN
‡CIBER in Bioengineering, Biomaterials & Nanomedicine (CIBER-BBN), SPAIN

***Abstract*—Electrophysiological simulations are computationally expensive tasks. This kind of simulations are usually run in super-computers or clusters, which may be expensive or difficult to access. In this work we present DENIS@Home, a simulation platform that follows the Volunteer Computing paradigm. DENIS@Home is based on BOINC, a volunteer computer environment used worldwide, and on CellML, an open standard for mathematical modeling. In this paper we describe the system and show its advantages over more traditional approaches, and also discuss its potential utility for the scientific community.**

I. Introduction

Mathematical models of the heart's electrical activity are a powerful tool in cardiac research. These electrophysiological models can be used to analyze what is happening inside the cardiac cells during a disease, when a drug is given, or in many other situations. From the earliest mathematical models, the complexity of recent electrophysiological models has grown considerably, and, consequently, the computational cost of simulations has also grown.

The overall trend in recent years is to simulate thousands of variations of the same electrophysiological model with the purpose of reflecting the variability exhibited in live-cell experiments, and of predicting better the variable cardiac behavior in different subjects. For example, in [1] the authors analyzed an initial population of 10,000 models to find 213 candidate models that are fully consistent with the experimental data. In [2], a genetic algorithm was used to get a more accurately parameterized model. They needed to simulate 100 generations of 500 individuals (a total of 50,000 simulations).

This kind of problems can be solved by using supercomputers, but access to these infrastructures can be really restrictive because of their huge acquisition and maintenance costs. In this work we present an architecture based on Volunteer Computing (VC) that helps to overcome that limitation. The DENIS project (*Distributed Computing, Electrophysiological Models, Networking Collaboration, In Silico Research, Sharing Knowledge*, http://denis.usj.es) can help groups of scientist who do not have access to a supercomputer to carry out their research.

In this paper we describe how the VC simulator that is inside the DENIS project, works, and how the project can run a vast number of simulations based on collaboratively-generated CellML models. Also, we demonstrate the capabilities of the project with a use case.

II. Background

DENIS is built upon two different projects. The first one is BOINC, a pioneering resource for volunteer computing that serves as basis for different projects. The other one is the CellML language, which is an open standard based on the XML markup language to store and exchange computer-based mathematical models. Both projects are summarized next.

A. Volunteer computing and the BOINC Project

Volunteer computing (VC) is a computational paradigm in which volunteers provide computing resources to projects. Projects make use of the volunteer's resources to perform computational operations requiring a huge quantity of computing power. In VC systems, the work is divided into smaller work units, and each work unit is sent to a different volunteer's computer (*a host*) to be carried out. In this way, by simultaneously running a number of units, the work can be completed much faster than in a dedicated machine.

The most extended VC platform is the *Berkeley Open Infrastructure for Network Computing* (BOINC) [3]. This platform provides researchers with a set of tools to coordinate the work, a web page to show the progress of the project, and forums that enable volunteers to contact the administrators and other volunteers. A lot of VC projects use BOINC as scheduler of their work, since it is specially useful to solve problems that need huge compute capacity. Some BOINC projects have an enormous list of volunteers like SETI@Home or EINSTEIN@Home than can handle thousands of hosts with computing power in the order of PetaFLOPS.

The problems to be solved by BOINC projects are divided into small parts that are sent to volunteers. This approach generates a server central architecture that coordinates an entire and powerful computer network with just an small server. To execute the software and coordinate work, each host of the network has to install the BOINC client. The client program is the same for all BOINC projects and allows volunteers to choose which projects they want to collaborate with. The BOINC client acts as an interface between the BOINC servers and the host. It downloads the correct executable version to

the host as well as the configuration files generated for each task. When the task is completed, it uploads the resulting file to the server and sends information about the task's status (for example, whether the simulation was aborted by errors, or if it could not be run for other reasons).

B. CellML

CellML language [4] is an open standard based on the XML markup language. It has been created to share computer-based mathematical models. The parts of a CellML model are: a) the information about the model structure (how it is organized), b) the equations describing the system using MathML, and c) metadata that contains additional information about the model, which allows researchers to search for model components or for the model itself in a database or repository.

One on the most useful features of CellML is the reusability. As pointed out by Gianni *et al.* [5], scientists can share variables between different models, which makes it easy to generate one model as an upgrade of other model. Also, CellML is based on XML, so it is not language- or platform-dependant, and by using its API models can be exported from CellML to different programming languages.

A growing community is using CellML to build and share computer-based physiological models. For example, the Physiome Model Repository(*PMR*) [6] contains more than 500 different CellML models ordered by category. DENIS uses CellML as standard for the input models. The large number of available models and all the advantages mentioned above make it an adequate choice for our project.

III. Related work

The need for computational power to run electrophysiological simulations has been addressed with different approaches, some of which are summarized next. The first versions of these simulation tools were run in regular desktop computers, although nowadays most of them try to parallelize the simulations by using clusters and GPUs.

OpenCOR [7] is an open source and cross-platform modeling environment based on COR. It's a desktop application that allows users to create, edit and simulate CellML Models. It has a big community behind it that modifies the software and uses it intensively. It's also based on OpenCell, which is a different environment of cell simulations. Its strongest point is the ease of use. However, it runs in stand-alone computers, which makes it useful only for small simulations.

EMOS [8] is a simulation software written in FORTRAN that uses MPI advantages to simulate electrophysiological models using different computers. This software simulates from isolated cells to the complete heart. It is a finite element code for the resolution of the monodomain equation based on the Operator Splitting algorithm. It has 11 different models encoded. The post-processing software generates output compatible with Ensight and Paraview visualization software.

CHASTE project [9], [10] also uses the CellML repository and traduces models to C++ code in order to execute simulations in different scales (from single cells to the complete heart). It offers numerous numerical methods to solve the ordinary differential equations in the models. As previous platforms, it is oriented to run in a single computer or in a cluster.

Myokit [11] is developed in Python, and uses thread control to enable the parallelization of the simulation inside the same computer, and OpenGL to parallelize in graphic cards (GPGPU). It is able to simulate from single cell to 2D tissues using an great variety of models from PMR.

Mena *et al.* [12] uses GPUs to parallelize the work. Their solutions can improve 180x the time to solve the simulation, making possible to simulate an entire heart.

The Cardiac Electrophysiology Web Lab [13] is an online tool for the characterization and comparison of electrophysiological cell models in a wide range of experimental scenarios. All the models used are coded in CellML and the system currently contains a sample of 36 models and 23 protocols, including current-voltage curve generation, action potential properties under steady pacing at different rates, restitution properties, block of particular channels, and hypo/hyperkalemia.

DENIS, our approach, also uses CellML to define the models, as most of these projects. However, the main difference with them is that DENIS can use an entire network of personal computers or clusters to perform the computations, by distributing the simulations between all the nodes in the network. This makes it possible to tackle complex simulation problems that are not feasible for stand alone computers.

IV. System overview

The architecture of the solution can be divided into two parts. The first one, named DENIS Simulator, is the application that runs into the hosts, and the second one, named DENIS@Home, is a collection of BOINC services that coordinates the distribution of the DENIS Simulator and sends the work and configuration files to the hosts.

A. DENIS Simulator

DENIS Simulator is the application that runs in the volunteer's computers (Fig. 1). This application simulates electrophysiological models described in CellML files under specific conditions. It is generated by using the CellML API and the BOINC API. DENIS Simulator needs to be compiled to different platforms in order to be executed in the heterogeneous computing network of volunteers.

1) CellML Exporter: In DENIS, CellML models are obtained from the PMR. Other models can be added to the DENIS Simulator, but they have to be tested before their addition. To include CellML models into the DENIS Simulator, a CellML exporter was developed to translate from CellML to C++. This tool generates a C++ library with all the models. This library is later called from the mathematical solver of *DENIS Simulator*.

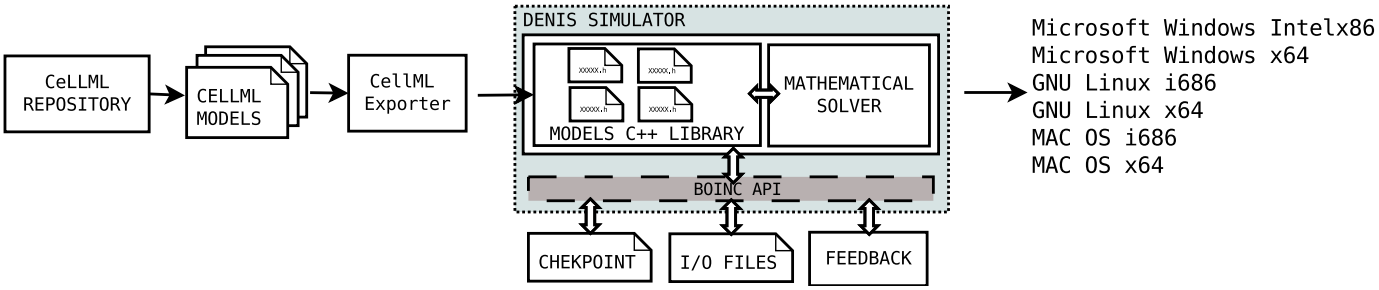

Fig. 1. Architecture of DENIS Simulator

2) Mathematical Solver of DENIS Simulator: DENIS Simulator uses the Forward-Euler numerical method to solve the electrophysiologycal model's ordinary differential equations. The solver collects the model's equations from the library generated with the CellML exporter.

3) BOINC API: The BOINC API provides different tools to abstract the application development from the host that performs the simulation. DENIS simulator uses the BOINC API to: a) store checkpoints, b) get input files and generate output files that the services of BOINC can recognize, and c) give feedback of the simulation progress to the volunteer.

Checkpoints are a critical part of DENIS Simulator, specially for long simulations. Volunteers may not share any resources with DENIS@Home at a given time, or may share them between different BOINC-based projects. The software stores the simulated data in a temporal file to avoid losing information if the task is stopped at the volunteer host.

4) Multi-Architecture Compilation: DENIS Simulator is compiled for different platforms, so that it can run in as many volunteer hosts as possible.

When a new version of the application is suitable to be released, it is compiled by using Virtual Machines running old versions of the most common Operating System to maximize the compatibility for volunteers hosts. At the moment, DENIS Simulator is compiled for 6 alternatives: 32 and 64 bits versions of GNU Linux, Microsoft Windows and MAC OS.

After compilation, each executable is software-signed. That prevents that attackers can distribute malicious software as an official version of DENIS Simulator. To avoid an external attack, the signature process is made in a live USB GNU Linux encrypted device, and the entire process is done without internet connection.

B. Simulations workflow

The sequence of steps necessary to complete a simulation is shown in Fig. 2. Simulations start with a configuration file. This file is filled by the scientist that wants the simulation done. It contains different parameters that are necessary to run the simulation: the model to be simulated, duration of the simulation, time step for the integration of the model, the output frequency, new value for constants of the model, and the output variables that will be stored in the output file.

Each job of DENIS@Home consists of a configuration file and the DENIS Simulator (Fig. 2.(1)). When a job is launched it generates a workunit (Fig. 2.(2)) that creates two tasks with the same configuration file (Fig. 2.(3)). These identical tasks are sent to two different volunteers. This redundancy is included to protect the results against hacking attempts or simulation problems. When the volunteers end the tasks, the BOINC client sends back the results. In step (4), the results of both volunteers are compared, and if they are not equal a third identical task is generated and sent to another volunteer. This process is repeated until at least two task produce the same results, with a limit of 10 retries. The third task will also be generated if one of the first tasks reaches the task deadline without finishing. The task deadline is the maximum allowed time between the reception of the task by the host and the sending of results.

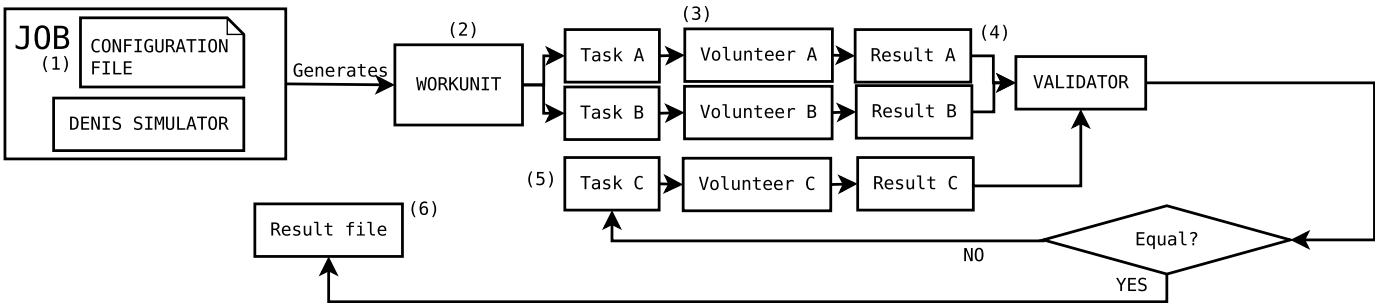

Fig. 2. Steps necessary to complete one simulation.

C. Architecture of DENIS@Home

All BOINC-based projects [3] consist of three main blocks: web server, task server and data server. In this section we describe how the general structure has been adapted for DENIS@Home.

1) Web server: allows the users to interact between them and the staff. It is useful for reporting bugs on the application or improvements to the DENIS Simulator or work distribution. The web server is also used by the BOINC client to request new tasks and send results of the finished tasks at the host.

2) Task server: consists of different services that organize and dispatch the work to the users. It greps the configuration files and the application and prepares them to be sent to the volunteers. This block is composed by the following elements:

a) Feeder: responsible for creating a workunit for each configuration file on the input folder. Each workunit is stored in the *Workunit System* which is responsible for generating the necessary tasks in order to obtain a valid result.

b) Validator: compares the result files received from different volunteers for the same workunit. The validation process is done by comparing 10 lines of the result files selected randomly. To check the entire document could cost so much time and this could overload the server. If they match as equal, the workunit is closed and marked as validated. If the results are different, one new task is created to test against them. Once a workunit is validated, the result files are sent to the *Assimilator*.

c) Assimilator: selects one of the matching result files and saves it into the *Results folder* in the *Data Server*. The result file is stored as a readable CSV file, that can be easily imported into a spreadsheet software. This file contains the time and the results of the variable that were marked as output in the configuration file.

3) Data server: consists of the different databases to store information about the state of the simulations and the storage systems for the configuration, result files, and the elecrophysiological markers.

a) Google Drive synchronization: Due to the vast amount of data that can be generated in DENIS, results storage is a critical point. If all data is stored in the server, it could collapse, and it is expensive to create and maintain a big storage system. To avoid that problem and to facilitate the scientist to get all their results files, we have connected DENIS with Google Drive.

Each scientist has a folder structure inside Google Drive with three folders: DENIS-Config, DENIS-ToSend, and DENIS-Results. They can generate the configuration files and store them in DENIS-ToSend. Periodically, the *Google Drive Synchronization* module collects the files from this folder and save them in the *Input folder*. The configuration files are also copied to the *DENIS-Config* folder in Google Drive to indicate that the simulation have been sent.

Once the result files are copied to the *Output folder* by the *Assimilator*, the *Google Drive Synchronization* module moves them to the DENIS-Results folder in Google Drive.

b) Workunit system: This module manages the workunits created by the *Feeder*. For each workunit, two tasks are created and sent to different volunteers in the network. In order to mark one workunit as finished, two tasks have to be completed with the same result. As described above, if the validator detects that the results are different, a new task is sent. This module also controls the tasks deadline (or *delay bound* as referred to in BOINC projects).

c) Database of electrophysiological markers: There are several electrophysiological markers that are always computed in cardiac simulations. In order to facilitate that scientists reuse and share them, the results are stored in a No-SQL database. This information can be also used to calibrate model populations for problems as the ones showed in the introduction of this paper.

V. DENIS@Home test case

Volunteer computing is commonly used to solve multi-dimensional problems, and to carry out highly parallelizable tasks. The study of different sized tasks is useful to see how the network works with varying tasks size and which ones are suitable to electrophysiological simulation.

In order to answer this question, we performed an experiment that consisted on sending blocks of simulations of different lengths. We then analyzed the time costs for the completion of the tasks in each block. Although the number keeps increasing steadily, DENIS@Home had 20,915 registered hosts when the experiment was performed,

A. Test Design

Groups of 10,000 tasks were simulated. Each group of tasks had a different simulation length: 200, 400, 600 and 1000 seconds of cell activity. The model selected for the experiment was the one proposed by Carro *et al.* [14] using a time step of 0.002 ms to solve the differential equations. Only the last second of each simulation were stored in the result file with and time step of 0.1 ms.

Each group of tasks got a different delay bound to accomplish the entire work: 4, 5, 6 and 10.5 days respectively. This time was selected based on the computational cost of each task for a normal user.

Two values were extracted from the DENIS@Home database to analyze the behavior of the different groups of tasks: the time to process each task and the time-difference between the creation of the workunit and the validation of the result.

B. Results

The collection data period was 1 month. During this month, 36 groups of 200 seconds, 31 groups of 400 seconds, 31 groups of 600 seconds and 20 groups of 1000 seconds were completely simulated. We discarded groups of tasks that had not been completed at the end of the month, and measured the average time to completion for the different groups of tasks.

As expected, the average time spent in the host (since tasks were received from the server until the simulations of the

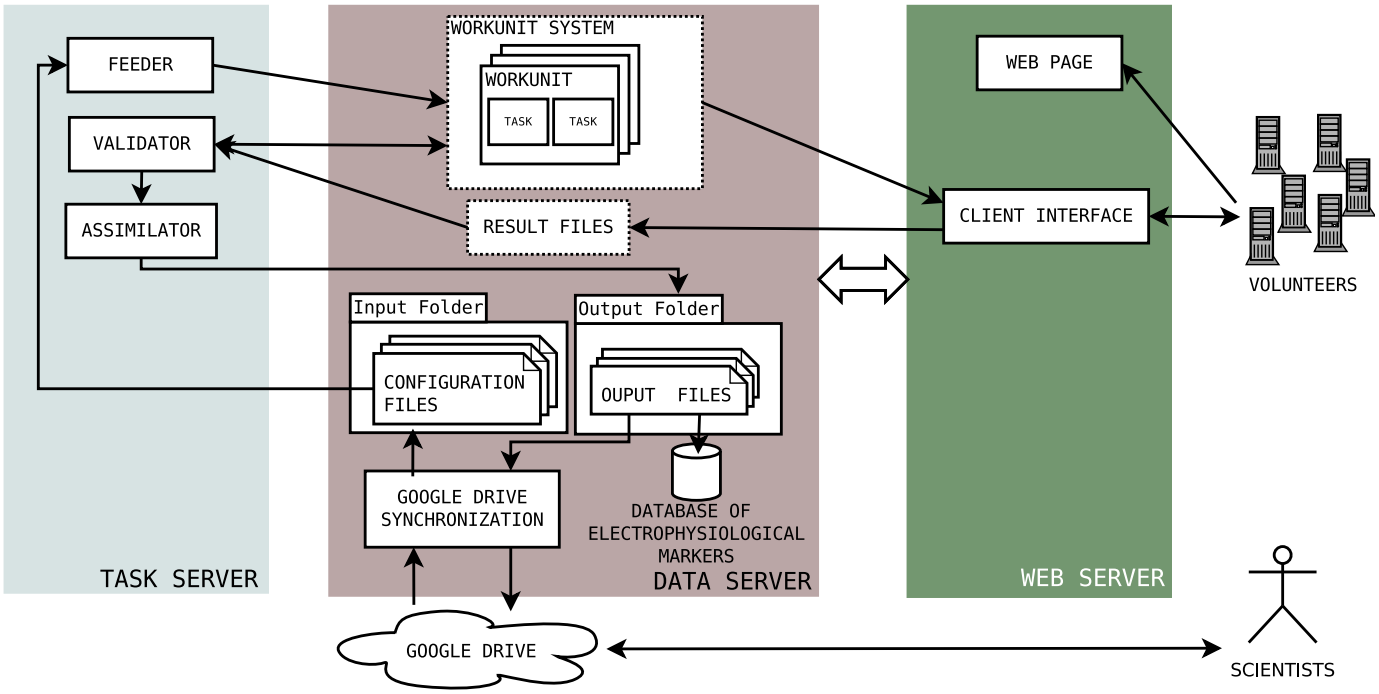

Fig. 3. DENIS@Home Architecture diagram

electrophysiological model finished) increased with the length of the simulation in most cases (see Fig. 4.(a)). The 50% of the 200-second-length tasks were completed in 1.58 hours, the 400-second-length tasks in 2.59 hours, the 600-second-length tasks in 4.27 hours, and the 100-second-length tasks in 8.30 hours. For a 75% of completion the difference between groups were smaller: the 200-second-length tasks were completed in 5.26 hours, of the 400-second-length tasks in 5.32 hours, of the 600-second-length tasks in 6.35 hours, and of the 100-second-length tasks in 13.73 hours.

The total average time spent in the system (since workunits were created until simulations of the electrophysiological model finished) increased with the length of the simulation, but for short simulation lengths differences were small (see Fig. 4.(b)).

VI. Discussion and conclusions

In this work we presented DENIS, a simulation platform based on BOINC and CellML. To demonstrate its capabilities we performed a test in which we measured the time needed to complete electrophysiological simulations of different lengths.

In Fig.4.(a), the first tasks to finish were the shortest ones. However, all tasks experienced delays and stops, so the time spent in the host was a sum of the computation time and idle time. In some cases the computation time for the shortest tasks was negligible in comparison with the idle time, which made the completion curve of the shortest tasks slow down for percentages over 70%.

Both in Figs.4.(a) and (b) there was a gap between short simulations (200, 400 and 600 seconds) and long ones (1000 seconds). In Fig.4.(a) this gap was due to differences in the length of the tasks, as was in the middle part of the curve of Fig.4.(b). The gap in the final part in Fig.4.(b) (percentages over 95%), however, seemed to be a consequence of the differences in the delay bound. When a volunteer did not finish a task within the allotted time, the task was sent to a different volunteer with a new deadline, which greatly increased the total completion time for that group of tasks.

As shown in Fig.4, simulation size had a relatively small effect on the time to completion of a group of tasks. This poses a great advantage over stand-alone computers, in which completion time increases proportionally with simulation length. Furthermore, the completion rate vs. total time increased steeply up to around 85%. This fact suggests that our system is specially suitable for exploratory studies in which preliminary results are needed in order to improve the model, such as model adjustment [1] or preliminary studies using approximated values of the markers in the database generated by DENIS.

DENIS@Home started as a novel BOINC project and has grown steadily since it started. At the beginning just a few beta testers collaborated with the project but in February 2016 there are 3492 volunteers collaborating with 29876 hosts. The computing capacity of DENIS@Home reached a peak of 97563.79 GigaFLOPS on October 2015. Such capacity makes it possible to tackle new simulation projects that were not viable in the past.

Acknowledgment

The authors would like to thank DENIS@Home volunteers for the support given to this project. Without the collaboration

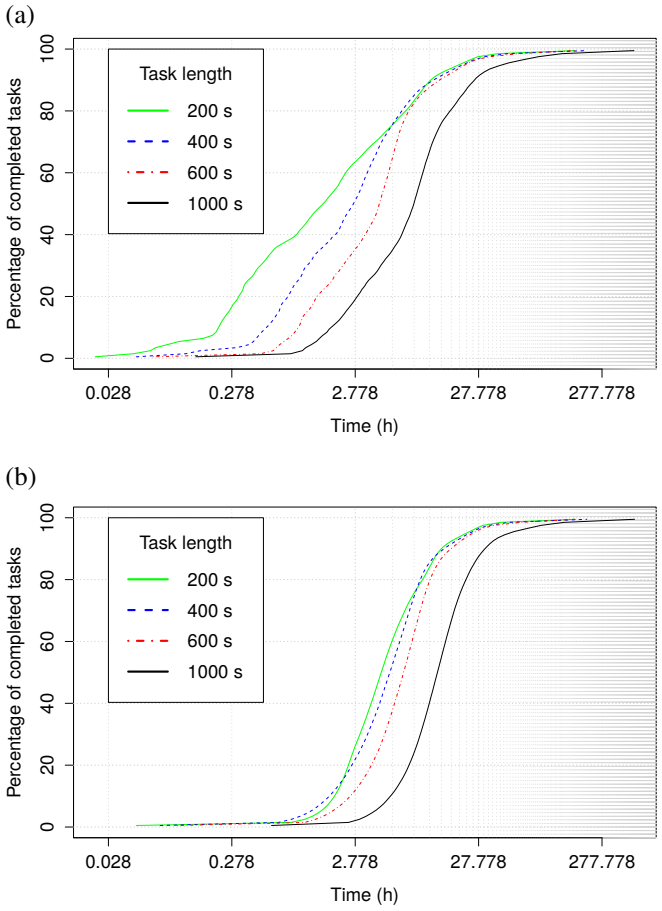

Fig. 4. Time to completion for different groups of tasks. (a) Percentage of completed tasks vs. average time spent in the host. (b) Percentage of completed tasks vs. average time from the creation of the workunit until the task finishes.

of all of them, collaborative projects like DENIS@Home would not be as useful for the science as they are.

This work was supported by CIBER in Bioengineering, Biomaterials & Nanomedicine (CIBER-BBN) through Instituto de Salud Carlos III, and by Grupo Consolidado BSICoS (T96) from DGA and European Social Fund.

REFERENCES

[1] O. J. Britton, A. Bueno-Orovio, K. Van Ammel, H. R. Lu, R. Towart, D. J. Gallacher, and B. Rodriguez, "Experimentally calibrated population of models predicts and explains intersubject variability in cardiac cellular electrophysiology," pp. E2098–105, jun 2013.

[2] W. Groenendaal, F. A. Ortega, A. R. Kherlopian, A. C. Zygmunt, T. Krogh-Madsen, and D. J. Christini, "Cell-Specific Cardiac Electrophysiology Models," *PLoS Comput Biol*, vol. 11, no. 4, p. e1004242, 2015. [Online]. Available: http://dx.doi.org/10.1371/journal.pcbi.1004242

[3] D. P. Anderson, E. Korpela, and R. Walton, "High-performance task distribution for volunteer computing," in *Proceedings - First International Conference on e-Science and Grid Computing, e-Science 2005*, ser. E-SCIENCE '05, vol. 2005. Washington, DC, USA: IEEE Computer Society, 2005, pp. 196–203. [Online]. Available: http://dx.doi.org/10.1109/E-SCIENCE.2005.51

[4] A. Cuellar, W. Hedley, M. Nelson, C. Lloyd, M. Halstead, D. Bullivant, D. Nickerson, P. Hunter, and P. Nielsen, "The CellML 1.1 Specification," *J Integr Bioinform*, vol. 12, no. 2, p. 259, 2015.

[5] D. Gianni, S. McKeever, T. Yu, R. Britten, H. Delingette, A. Frangi, P. Hunter, and N. Smith, "Sharing and reusing cardiovascular anatomical models over the Web: a step towards the implementation of the virtual physiological human project," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 368, no. 1921, pp. 3039–3056, 2010. [Online]. Available: http://rsta.royalsocietypublishing.org/content/368/1921/3039

[6] T. Yu, C. M. Lloyd, D. P. Nickerson, M. T. Cooling, A. K. Miller, A. Garny, J. R. Terkildsen, J. Lawson, R. D. Britten, P. J. Hunter, and P. M. F. Nielsen, "The physiome model repository 2," *Bioinformatics*, vol. 27, no. 5, pp. 743–744, mar 2011. [Online]. Available: http://bioinformatics.oxfordjournals.org/cgi/content/long/27/5/743

[7] A. Garny and P. J. Hunter, "OpenCOR: a modular and interoperable approach to computational biology," *Frontiers in physiology*, vol. 6, p. 26, 2015. [Online]. Available: http://europepmc.org/articles/PMC4319394

[8] E. A. Heidenreich, J. M. Ferrero, M. Doblaré, and J. F. Rodríguez, "Adaptive Macro Finite Elements for the Numerical Solution of Monodomain Equations in Cardiac Electrophysiology," *Annals of Biomedical Engineering*, vol. 38, no. 7, pp. 2331–2345, 2010. [Online]. Available: http://dx.doi.org/10.1007/s10439-010-9997-2

[9] J. Cooper, R. Spiteri, and G. Mirams, "Cellular cardiac electrophysiology modelling with Chaste and CellML," *Frontiers in Physiology*, vol. 5, p. 511, 2015. [Online]. Available: http://journal.frontiersin.org/Journal/10.3389/fphys.2014.00511

[10] G. R. Mirams, C. J. Arthurs, M. O. Bernabeu, R. Bordas, J. Cooper, A. Corrias, Y. Davit, S.-J. Dunn, A. G. Fletcher, D. G. Harvey, M. E. Marsh, J. M. Osborne, P. Pathmanathan, J. Pitt-Francis, J. Southern, N. Zemzemi, and D. J. Gavaghan, "Chaste: An Open Source C++ Library for Computational Physiology and Biology," *PLoS Comput Biol*, vol. 9, no. 3, p. e1002970, 2013. [Online]. Available: http://dx.doi.org/10.1371/journal.pcbi.1002970

[11] M. Clerx, P. Collins, E. de Lange, and P. G. Volders, "Myokit: A simple interface to cardiac cellular electrophysiology," *Progress in Biophysics and Molecular Biology*, dec 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0079610715002576

[12] A. Mena, J. M. Ferrero, and J. F. Rodríguez Matas, "GPU accelerated solver for nonlinear reaction–diffusion systems. Application to the electrophysiology problem," *Computer Physics Communications*, vol. 196, pp. 280–289, nov 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010465515002635

[13] J. Cooper, M. Scharm, and G. R. Mirams, "The Cardiac Electrophysiology Web Lab," *Biophys. J.*, vol. 110, no. 2, pp. 292–300, 2016.

[14] J. Carro, J. F. Rodríguez, P. Laguna, and E. Pueyo, "A human ventricular cell model for investigation of cardiac arrhythmias under hyperkalaemic conditions," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 369, no. 1954, pp. 4205–4232, 2011. [Online]. Available: http://rsta.royalsocietypublishing.org/content/369/1954/4205