Parallel stochastic systems biology in the cloud

Parallel stochastic systems biology in the cloud

Journal:	*Briefings in Bioinformatics*
Manuscript ID:	BIB-13-0006.R2
Manuscript Type:	Paper
Date Submitted by the Author:	19-May-2013
Complete List of Authors:	Aldinucci, Marco; University of Torino, Computer Science Department Torquati, Massimo; University of Pisa, Computer Science Department Spampinato, Concetto; University of Catania, Department of Electrical, Electronics and Computer Engineering Drocco, Maurizio; University of Torino, Computer Science Department Misale, Claudia; University of Torino, Computer Science Department Calcagno, Cristina; University of Torino, Department of Life Sciences and Systems Biology Coppo, Mario; University of Torino, Computer Science Department
Keywords:	stochastic simulation, cloud, multi-core, systems biology, Parallel patterns, Gillespie algorithm

Parallel stochastic systems biology in the cloud

Marco Aldinucci, Massimo Torquati, Concetto Spampinato, Maurizio Drocco, Claudia Misale, Cristina Calcagno, Mario Coppo

Marco Aldinucci, Maurizio Drocco, Claudia Misale, Mario Coppo are with the Computer Science Department, University of Torino, Corso Svizzera 185, 10149 Torino, Italy.
E-mail: marco.aldinucci@unito.it, maurizio.drocco@gmail.com, claudia.misale@unito.it, mario.coppo@unito.it

Massimo Torquati is with the Computer Science Department, University of Pisa, Largo B. Pontecorvo 3, 56127 Pisa, Italy
E-mail: torquati@di.unipi.it

Concetto Spampinato is with the Department of Electrical, Electronics and Computer Engineering, University of Catania, Viale Andrea Doria 6, 95125 Catania, Italy
Email: cspampin@dieei.unict.it
Cristina Calcagno is with Department of Life Sciences and Systems Biology, University of Torino, Viale P. A. Mattioli 25, 10125 Torino, Italy
E-mail: cristina.calcagno@unito.it

Abstract

The stochastic modelling of biological systems, coupled with Monte Carlo simulation of models, is an increasingly popular technique in bioinformatics. The simulation-analysis workflow may result computationally expensive reducing the interactivity required in the model tuning. In this work we advocate the high-level software design as a vehicle for building efficient and portable parallel simulators for the cloud. In particular, the Calculus of Wrapped Components (CWC) simulator for systems biology, which is designed according to the FastFlow pattern-based approach, is presented and discussed. Thanks to the FastFlow framework, the CWC simulator is designed as a high-level workflow that can simulate CWC models, merge simulation results and statistically analyse them in a single parallel workflow in the cloud. To improve interactivity, successive phases are pipelined in such a way that the workflow begins to output a stream of analysis results immediately after simulation is started. Performance and effectiveness of the CWC simulator are validated on the Amazon Elastic Compute Cloud.

Keywords: Stochastic simulation; cloud; multi-core; distributed computing; parallel patterns.

Key points:

- A methodological study on the design of a parallel, cloud-enabled simulator-analysis pipeline for systems biology, which is paradigmatic example of a broad class of algorithms for bioinformatics.
- Portability and performance portability, from multi-core to the cloud, of parallel applications via high-level design and pattern-based development framework.
- An extensive survey of the methods and the tools for bioinformatics on the cloud and the methodologies for their development with a specific focus on high-level approaches supporting easy engineering, performance and portability.

1 Introduction

The stochastic simulation of biological systems is an increasingly popular technique in bioinformatics, as either an alternative or a complementary tool to traditional differential equations (ODEs) solvers. This trend, starting from Gillespie's seminal work [1], has been supported by a growing number of formalisms aiming to describe biological systems as stochastic models [2]. The stochastic modelling approach is computationally more expensive than ODEs. Nevertheless, it is still considered attractive for its superior ability to describe transient behaviours of biological systems, e.g. divergent trends and spikes that are typically hidden in the averaged process described by ODEs. The stochastic modelling typically relies on the Monte Carlo method, which is also used in related domains of systems biology, e.g. epidemiology and phylogeny [3, 4].

The high computational cost of stochastic simulations is well known and has led, in the last two decades, to a number of attempts to accelerate them up by using several kinds of techniques, such as approximate simulation algorithms and parallel computing. In this work, this latter approach is taken into account exploiting an Infrastructure-as-a-Service (IaaS) in the cloud as a parallel execution environment.

Monte Carlo methods require the computation of many independent instances either to achieve statistically meaningful results or sensitivity analysis (via parameter sweeping). In both cases, these independent instances have been traditionally exploited in an *embarrassingly parallel* fashion, executing a partition of the instances (bag of tasks) on different platforms, and more recently on many-core GPGPU platforms [5]. This approach has been often coupled with High Performance Computing (HPC) infrastructures, such as grid or clusters of multi-/many-core. However, it suffers from some drawbacks related to design, performance, and usability of simulation tools.

Traditional HPC platforms are expensive to deploy (and rent); their configuration is hardly customisable. Moreover, HPC platforms suffer from reduced interactivity and might induce slow time-to-solution. Each experiment requires to enqueue the simulations in a shared environment, deploy initial data, simulate the model, gather results from a distributed environment, post-process them (often sequentially) and then eventually access the results. This process is typically repeated several times to fine-tune simulation parameters.

A further issue that deteriorates the reactivity and time-to-solution is the "sequentialisation" of simulation and analysis phases, which slow down the modelling-to-result process during the tuning of the biological model. The filtering and the analysis of raw results, which require the merging of data obtained from different simulation instances and their statistical analysis, is often demoted to a secondary aspect in the computation and treated with off-line post-processing tools, and frequently not even disclosed in performance results.

The very same approach is used also in recent efforts exploiting GPGPUs [6]. The ever-increasing size of produced data makes this approach no longer viable. As a solution, we advocate the offloading of the *whole* simulation-analysis process in the cloud as a single parallel pipeline with no storage of intermediate results on virtualised storage [7]. In this vision, data analysis is managed as an on-line process working on (high-frequency) streams of data resulting from the on-going simulations. This approach has non-trivial effects on tool design since both the parallel simulator and the parallel analysis should work on (high-frequency) streams, and require efficient data dependencies management (both on distributed and shared-memory systems). While the Monte Carlo simulation "in insulation" is an embarrassingly parallel process, the whole simulation-analysis workflow is not [8, 9].

Cloud technology carries the potential to overcome most of the above issues. It makes available on-demand and on a *pay-per-use* basis an elastic parallel computing platform that can be customised with a specific set of tools such as simulators for systems biology. Cloud "elasticity" enables the users to deploy the same application on a virtualised parallel platform of configurable type, size and computational power. The typical platform can be abstracted as a virtual cluster of shared-memory multi-core platforms. Once deployed, the virtualised platform is immediately ready to compute and can be interactively used by the end user. This potentiality, however, can be fully exploited only if the running software (e.g. simulation tool) exhibits a similar flexibility and interactivity:

- The application should benefit from both levels of parallelism available in a (virtualised) cluster of multi-core (and many-core if available), hopefully providing the end user with performance scalability with respect to both levels;

- The programming model should manage parallelism as a first-class concept to make the tools (e.g. simulator) easy to design, develop and extend; the programming model should be able to capture parallelism at all levels, i.e. distributed platforms, multi-core, and many-core, and possibly it should be able to support the seamless porting of application across the described platforms with performance portability.
- The software tools itself should be designed to be reactive and interactive in order to be dynamically steered by bioinformatics scientists.

This paper presents the parallelisation of stochastic processes in the light of virtualised distributed cluster of multi-core platforms and tools that are required to derive an efficient simulator from both performance and easy engineering viewpoints. The presented methodology makes it possible to run the same code from multi-core to virtualised cluster of multi-cores (i.e. private and public clouds infrastructures). This latter will be a key factor for the next generation of biological tools, since bioinformatics scientists are more interested in the accurate modelling of natural phenomena rather than on the low-level protocols required to build efficient tools on both multi-core platforms and large distributed execution environments.

In this work, the simulator for a stochastic calculus for systems biology, i.e. Calculus of Wrapped Compartments (CWC) [10], will be used as test-bed. The CWC simulator [7, 9], previously targeting multi-core platforms only, has been designed exploiting a high-level methodology based on parallel patterns and considering the whole simulation workflow: from simulation to on-line data analysis and mining. This solution can provide bioinformatic scientists with immediate feedback on simulation results and their main statistic estimators while the simulation is still running, thus with an early feedback on simulation effectiveness. The advocated high-level allows the design of the CWC simulator as a workflow of successive stages, where edges among stages are data dependencies. The exploitation of parallelism on multi-core, cluster and eventually in the cloud is almost entirely in charge of the parallel programming methodology provided by the FastFlow parallel programming framework [11, 12].

Although the parallelisation of stochastic simulators has been extensively studied in the last two decades [13], the main contributions of our work with respect to the state of the art are:

1. Addressing cloud IaaS-specific parallelisation issues;
2. Advocating a general parallelisation schema rather than a specific simulator;
3. Addressing the on-line data analysis, thus it is designed to manage data (possibly big data) in the cloud.

To the best of our knowledge, many related works cover some of these aspects, but none of them address all three aspects at the same time.

The remainder of the paper is structured as follows: in Sec. 2 related works is discussed along three main directions: methods and tools for developing cloud applications (Sec. 2.1), existing cloud-enabled bioinformatics applications (Sec. 2.2) and theoretical tools for medullisation in systems biology (Sec. 2.3). In Sec. 3 and 4 the FastFlow programming framework and the design of the CWC simulator are presented, respectively. In Sec. 5 tool implementation is validated against effectiveness and performance obtained on the Amazon EC2 public cloud infrastructure. Section 6 concludes the paper.

2 Background and related work

2.1 Developing applications for the Cloud

The cloud encompasses a *pay-per-use* business model. End users are not required to take care of hardware, power consumption, reliability, robustness, security, and the problems related to the deployment of a physical computing infrastructure. In IaaS cloud usage, the aggregate computing power and storage space are provided to user applications in an elastic fashion. In principle, they can be scaled up and down according to user needs, and billed accordingly. Applications running in an IaaS cloud are required to be scalable and designed to efficiently exploit a virtualised parallel platform, possibly exploiting different parallel programming models, e.g. shared-memory and message-passing. Alternatively, cloud technology can be exploited in the Software-as-a-Service (SaaS) fashion by exposing applicative services (rather than a platforms) to end-users. In the SaaS model, cloud elasticity is typically managed by domain-specific applications (or applicative frameworks) that exposes domain-specific services to end-users. The pay-per-use business model is typically applied in term of the Quality of Service (e.g. performance, latency, storage space) provided by the service.

In both cases, from developer viewpoint building a cloud-enabled application (or a service) is not easier than building a distributed application for cluster of multi-core platforms, which is well known to be a complex work.

At the present time, the aggregate computational power of on-demand virtualised cluster did not have reached the figures possible in massively parallel platforms[1]. Cloud technology cannot being currently considered suitable to target very high computational power needs.

[1] As an example, in spring 2013, the Amazon EC2 maxium core count for on-demand clusters is 160: 20 extra-large instances, each of them with 8 virtualised cores.

However, the shift toward cloud technology has many drivers that are likely to sustain this trend for several years to come. This is likely to make parallel computing methodologies increasingly accessible to a wider range of developers. Software technology is consequently changing: in the long term, writing parallel programs that are efficient, portable, and correct must be no more onerous than writing sequential programs. Such a transition will likely entail a significant raise of the level of abstraction of parallel programming models with respect to the current state-of-the-art in order to support the mainstream of software development, where human productivity, total cost and time to solution are equally, if not more, important than application performance.

Pragmatically, designing an application for the cloud requires dealing with the heterogeneous nature of the virtualised platforms in term of programming models needed to effectively exploit parallelism among virtualised cores of the same virtual machine and different virtual machines (and possibly different cloud deployments). Virtualised or not, shared memory multi-cores and clusters require different techniques and tools to support efficient parallelism exploitation. The *de facto* standard tools in these cases are OpenMP [14] and MPI [15], respectively. OpenMP can be considered a high-level of abstraction programming framework (at least for data-parallelism) but targets only shared-memory platforms. MPI, which is mainly exploited in distributed platforms, exhibits a rather low level of abstraction because it expose to developers the full complexity of a message-passing programming model. Applications developed with MPI often requires to be entirely re-designed to accommodate communication primitives that require to be fully interweaved with business code. OpenMP and MPI can be used conjunction to target clusters of shared-memory platforms; the integration of two programming models is fully in charge of the application developers.

Algorithmic skeletons approach (a.k.a. pattern-based approach) aims at reducing the development complexity of parallel software design by providing developers with a higher level of abstraction aiming to move most of the complexity due to communication/synchronisation management from developers to the programming framework [16, 17]. The algorithmic skeleton community has proposed various programming frameworks, aimed to provide the application programmer with quite high-level abstractions encapsulating parallelism exploitation patterns [18]. Some of them provide programmers with a higher level of abstraction, but are oriented to coarse grain computations (e.g. *ASSIST* [19], *StreamIt* [20], *Brook* [21]); some others target shared-memory platforms only (e.g. *Intel TBB* [22]). Not many of them provide a single programming models targeting heterogeneous environments such as clusters of multi-core platforms, which are the kind of platforms typically offered by IaaS cloud technology. One of them is the FastFlow framework, which will be presented in Sec. 3.

A popular specialisation of the pattern-based approach is represented by the *MapReduce* programming model [23], in which a single powerful pattern composed by the *pipelining* of *Map* and *Reduce* patterns is offered to the programmers (often also at level of cloud programming API). Being designed by Google for distributed computing to process data on a distributed file system on clusters, it might be not general enough to naturally cover all application needs (e.g. streaming, processing in memory, recursive computations) and deployment scenarios (e.g. virtualized multi-core).

The high-level parallel programming approach, based on algorithmic skeletons, has been already exploited in computational biology for distributed environments and grids. As an example, the MACACO simulator is realised as a pipeline of simulation and post-processing, where the simulation phase is realised in parallel by farming out the parameter sweeping of the stochastic simulation of calcium currents [24]. An orthogonal approach aimed to raise the level of abstraction of computation in bioinformatics is via Problem Solving Environments (PSE) and Domain-Specific Languages (DSL). As an example, in [25] a framework for morphogenesis simulation is presented.

2.2 Bioinformatics in the Cloud

The cloud has the potentiality to become an enabling technology for bioinformatics and computational biology. It can seamlessly provide applications and their users with large amount of computing power and storage in an elastic and on-demand fashion. This naturally meets the need of simple availability of processing large amount of heterogeneous data, of storing massive amount of data and of using the existing tools in different fields of bioinformatics. The ability of managing the whole data set in the cloud, as we advocate in this work, has been widely recognized as necessary for next generation bioinformatics [26]. As an example, the typical workflow of DNA sequencing [27] foresees that biologists design the experiments and send samples to sequencing centres, which make available raw data (through specific services, such as FTP, HTTP) to biologists, who have to download and use terabytes of data. At the same time, biologists copy the data into local machines for being used by bioinformatics scientists for the subsequent data analysis. This typical workflow implies that large (possibly big) amount data are moved several times from sites to sites, thus slowing down the analysis and the interpretation of the results. These multiple data movements can be partially or entirely avoided by moving the whole workflow in the cloud.

DNA sequencing and sequence alignment are classic examples of computational biology application in which having computing power as more as possible is never enough [28]. Examples of these applications are: *Crossbow* [29], a software pipeline for genome resequencing analysis which runs in the cloud (according to a MapReduce paradigm [30]) on top of *Hadoop* [31]; *CloudBurst* [32], which accomplishes mapping of next-generation sequence data to the human genome for a variety of biological experiments (e.g. SNP discovery and genotyping) achieving a significant speedup with respect to sequential execution, and *Myrna* [33], a differential gene expression calculation tool in large RNA-Seq datasets that integrates all the RNA sequencing steps (read alignment, normalisation, aggregation and statistical modelling) in a single cloud-based computational pipeline.

DNA sequencing is not the only bioinformatics application field for which the cloud has been adopted. Another example is in-silico organ modelling, which is a relatively new method for studying the development and functionality of human, and not only, body parts with computers. In [34], for example, a model of the human liver is emulated in a cloud-based system where each liver lobule is represented by Monte Carlo samples. By using this architecture, the authors demonstrate that the parallel computing paradigm permits to develop systems emulating organs with functionalities equivalent to those of an in-vitro specimen. A multi-scale model for the progression of pancreatic cancer that can be executed on a cloud platform is presented in [35]. This platform is designed for the needs of life science and pharmaceutical research allowing the integration of physiologically based and classical approaches to model drug pharmacokinetics and pharmacodynamics as well as metabolic and signalling networks.

Protein folding simulation is another notable example of calculation intensive process. In particular, it is the process that converts a two-dimensional unfolded polypeptide in a three dimensional structure. *Folding@home* initiative [36] attempted to attack the problem via opportunistic computing, distributing tasks to Internet users. Although *Folding@home* can be executed on a multitude of hardware platforms, given the unreliability of internet computer clients, the performance of the project is hindered by errors in the local network or the computers themselves wasting, otherwise useful, computer resources. The *Microsoft@home* project permits the execution of generic scientific computer-intensive applications, including *Folding@home*, in the cloud.

Simulation modelling of biological processes is the backbone of systems biology and discrete stochastic models are particularly effective for describing molecular interaction at different levels [37]. Nevertheless, it is common knowledge that these types of stochastic simulations, as for instance the Monte Carlo ones, are computationally intensive, and among the bioinformatics applications they are the ones that could benefit from distributed implementations on the cloud.

Despite the evident advantages of carrying out simulations on the cloud, at the moment, cloud based simulators occur at a slow pace and the scientific community is not fully exploiting the opportunity to grasp the potential of the cloud paradigm. While implementations and services for Monte Carlo simulations on the cloud [35, 38, 39, 40, 41] are few, the convenience of having virtually unlimited resources will make the cloud platform the perfect candidate for calculation-intensive applications.

Table 1 compares the features of some computational biology and bioinformatics tools freely available on the web.

[Table 1]

From a more general perspective, given that many existing bioinformatics tools and simulators rely on web services, their transition to a cloud based infrastructure will be quite natural and we expect, in the near future, that cloud-based bioinformatics applications and services will be created at accelerating pace. Examples of such a transition, which have been already put in place, are *CloudBurst* [32] which (as above described) maps next generation sequencing data [42] and *Cloud Blast*, a "clouded" implementation of NCBI BLAST [43], which basically have kept the same web service based architecture but changed the underlying hardware infrastructure to a cloud based one.

Cloud computing, however, poses a few "still unsolved" problems both for developers and users of cloud based software, ranging from data transfers over low-bandwidth networks to privacy and security issues. These aspects lead to inefficiency for some types of problems and future solutions should address such issues [27].

2.3 Calculi and Tools for Bioinformatics

In the field of biological modelling, tools such as *SPiM* [44], *Dizzy* [45], and *Bio-PEPA Workbench* [46] have been used to capture first order approximations to system dynamics using a combination of stochastic simulation and ODE approximation. These tools mainly target single processor boxes and, to the best of our knowledge, do not target distributed systems and cloud.

The Swarm algorithm [47], which is well suited for biochemical pathway optimisation, has been used in a distributed environment, e.g. in *Grid Cellware* [48], a grid-based modelling and simulation tool. *DiVinE* is a general distributed verification environment meant to support the development of distributed enumerative model checking algorithms including probabilistic analysis features used for biological systems analysis [49].

StochKit [50] is a C++ stochastic simulation framework implementing the Gillespie algorithm that targets multi-core platforms in its second version. It does not implement on-line trajectory reduction but it is performed in a post-processing phase. A first form of on-line reduction of simulation trajectories has been experimented within *StochKit-FF* [8], which is an extension of StochKit using the FastFlow runtime.

In [51] a parallel computing platform has been employed to simulate a large biochemical network using the Gillespie algorithm on multiple processors. The analysis of the simulation results to characterise the intrinsic noise of the network is done as a post-processing step.

Hy3S software package [52], that includes hybrid stochastic simulation algorithms, and *SRSim* [53], that performs rule-based spatial modelling, are both embarrassingly parallelised by way of the MPI library. *StochSimGPU* [54] exploits GPGPU for parallel stochastic simulations of biological systems. The tool allows computing averages and histograms of the molecular populations across the sampled realizations on the GPGPU. The tool is built on top of a GPGPU-accelerated version of the Matlab framework.

A schematic comparison of the main features of the biological simulation tools cited above is reported in Table 2.

[Table 2]

3 The FastFlow programming framework

[Figure 1]

FastFlow [55] is a structured parallel programming framework originally designed for shared-memory multi-core/many-core platforms. It has been recently extended to support distributed systems and cloud [12]. FastFlow provides programmers with predefined and customisable parallel design patterns (a.k.a. algorithmic skeletons) [16, 18]. They include stream-oriented patterns (*farm, farm-with-feedback, pipeline)* and data-parallel patterns *(map, reduce)*. Patterns can be arbitrarily composed to express higher-level patterns, e.g. *MapReduce, Divide&Conquer* [11].

FastFlow design is layered (see Figure 1); each layer is implemented on to of the next layer down. From top to bottom:

- *Parallel patterns* are in the top layer. It provides to the application programmers patterns abstractions that can be used on all platform families: multi-cores, many-cores, distributed systems and clouds. Once applications have been developed using patterns, it can be seamlessly ported across different platform families without re-designing or re-coding the application.
- *Arbitrary streaming networks* layer provides basic abstractions to implement each pattern as a data-flow graph [56], i.e. nodes and channels. A node (so-called `ff_node`) implements the basic unit of parallelism and can be used to encapsulate a sequential computation or a channel mediator, which are used either to build

collective communications possibly with user-defined semantics, both in the shared-memory or message-passing programming models.

- *Simple streaming networks* layer provide low-latency zero-copy single-consumer-single-producer channels for both shared-memory and message-passing programming models. The shared-memory channel is implemented via FIFO wait-free queues (FF-SPSC) [57]; the message passing support is built on top of *ZeroMQ* channels [58].

Overall, the FastFlow run-time support is realized as a header-only C++ template library and it is based on streaming of pointers, which are used as synchronisation tokens. This abstraction is kept also in the distributed version (i.e. across network channels) where data is transferred across network links by way of zero-copy channels. We refer back to [55] for further details.

4 The CWC multi-core simulator

4.1 The Calculus of Wrapped Compartments

In this work, the simulator for the Calculus of Wrapped Compartments (CWC) will be used as test-bed. CWC [59, 10] is a recently proposed formalism, based on term rewriting, for the representation of biological systems. CWC extends the usual representation of biochemical systems with reaction rules by adding a nested structure of labelled compartments delimited by membranes. However, to better focus on the proposed methodology and make the paper self-contained, we will write all examples of the paper in the basic subset of CWC, in which biochemical reactions are denoted in a standard, self-explanatory way. We only remark that in CWC a reaction is associated with a rate function depending on the overall content of the ambient in which the reaction takes place. This allows tailoring the reaction rates on the specific characteristics of the system, as for instance when representing nonlinear reactions as Michaelis-Menten kinetics. This function is simply represented by the kinetic constant for reaction whose rate is determined by the usual mass action law. We refer to [59, 10] for a complete presentation of CWC.

4.2 The CWC simulator

The CWC simulator [60] is an open source tool that implements Gillespie's algorithm on CWC terms. It handles CWC models with different rating semantics (law of mass action, Michaelis-Menten kinetics, Hill equation) and it can run independent stochastic simulations. The CWC simulator was designed using the FastFlow high-level methodology and targets multi-core platforms. It exploits both parallel simulation and data analysis in a single workflow. To make it possible, all the logical phases of the process (i.e. data distribution, parallel simulations, result gathering, parallel trajectory, data assembling and analysis) must be effectively pipelined. This implies that all phases work on a data streams.

The simulation workflow is composed of a three-stage pipeline: simulation, analysis, and display of results. The former two stages are in turn pipelines, whereas the display of results is realised by way of a Graphical User Interface (GUI).

4.2.1 The simulation pipeline

The simulation pipeline is composed of three main parts: a *generation of simulation tasks* stage, a *farm of simulation engines* stage and an *alignment of trajectories* stage [9].

The input of the simulation pipeline (either from GUI or from file) contains the model to be simulated and the parameters of the simulation. The output is a stream of arrays of simulation results. Each of these arrays holds a point for each of the trajectories of all (independent) simulations, aligned at a given simulation time. Actually, each array represents a snapshot (called “cut”) at a given simulation time of the whole dataset of results. This not necessarily represents the current status (at a given point in wall-clock time) of all running simulations. Stochastic processes exhibit an irregular behaviour in space and time according to their nature, since different simulations may cover the same simulation timespan, following many different (randomly-chosen) paths, in a different number of iterations. Therefore, parallelisation tools should support the dynamic and active balancing of workload across the involved cores. This mainly motivates the structure of the simulation pipeline. The first stage generates a number of independent simulation tasks, each of them wrapped in a C++ object.

These objects are passed to the farm of simulation engines, which dispatch them (on-demand) to a number of simulation engines (sim eng). Each simulation engine brings forward a simulation that lasts a precise simulation time (simulation quantum). Then it reschedules back the operation along the feedback channel. Simulation results produced in this quantum are streamed toward the next stage, which sorts out all received results and aligns them according to the simulation time. Once all simulation tasks overcome a given simulation time, an arrays of results is produced and streamed to the analysis pipeline.

In this process, the farm scheduler prioritises “slow” simulation tasks, in such a way that the front-line task proceeds as much aligned as possible to simulation time. This solves both the load-balancing problem by keeping all simulation engines always busy and reduces to the minimum the transient storage of incomplete results, thus reducing the shared-memory traffic.

[Figure2]

4.2.2 The analysis pipeline

By design, each cut of simulation trajectories (i.e. an array of simulation results), can be analysed immediately and independently (thus concurrently) from each other. For example, the mean and variance (as well as other statistical estimators) can be immediately computed and streamed out to the display stage. More complex analysis, i.e. ones aimed to understand system dynamics, has further requirements. In the most general case, they require the access to the whole dataset. Unfortunately, this can be hardly done with a fully on-line process. In many cases it is possible to derive reasonable approximation of these analysis from a sliding window of the whole dataset. For this reason, stream incoming in the analysis pipeline is passed through a stage that creates a stream of (partially overlapping) sliding windows of trajectories cuts. Each sliding window is processed in parallel and therefore is dispatched to a farm of statistic engines. Results are collected and re-ordered (i.e. gathered) and streamed toward user interface and permanent storage [7].

4.2.3 The graphical user interface

The CWC simulation-analysis pipeline is wrapped in a back-end tool that can be steered either via command line tools or a graphical user interface, which makes it possible to design the biological model, run simulations and analysis and to view partial results during the run. Also, the front-end allows controlling the simulation workflow from a remote machine.

The overall architecture of the CWC distributed simulator is shown in Figure 2. The first stage of the pipeline (simulation pipeline) has been implemented using a task-farm parallel paradigm where each simulation pipeline can be run on a different node in a cluster or cloud environment. It receives simulation parameters from the *generation of simulation tasks* node, and feeds the *alignment of trajectories* node with a stream of results.

In Figure 3 is presented a screenshot of the graphical user interface, in which the user has the possibility apply different kind of statistical analysis to data resulting from simulation.

[Figure 3]

5 Experimental evaluation

The evaluation of the CWC simulator takes into account the performance on both a single multi-core virtual machine and a small cluster of multi-core virtual machines running in the cloud. The ability of the CWC formalism to describe simple but significant biological systems, together with the effectiveness of the proposed on-line analysis to capture the behaviour of the system has been discussed in previous works [7, 10]. A simple yet significant example is reported in the following.

5.1 Simulation and analysis of an oscillatory system

As an example, consider the theoretical model for circadian oscillations based on transcriptional regulation of the frequency (frq) gene in the fungus *Neurospora* [61]. In this system the only specific feature of CWC used is the possibility of evaluate reaction rates introducing functions depending on time as well on the overall state of the system. For the sake of simplicity, we represent the model using a standard syntax for biochemical reactions [1]: when a reaction is decorated with a number it is understood that its rate is determined by the mass action law and then the number decorating it represents the kinetic constant of the reaction.

The model relies on the feedback exerted on the expression of the frq gene by its protein product FRQ. In this model, sustained rhythmic variations in protein and mRNA (M_{cyto}) levels occur, in the form of limit cycle oscillations [61]. The CWC rules[2] modelling this case are:

$$M_{cyto} \xrightarrow{0.5} M_{cyto}\ FRQ_{cito} \qquad FRQ_{nucleus} \xrightarrow{f_{FRQ}(t)} FRQ_{nucleus}\ M_{cyto}$$

$$M_{cyto} \xrightarrow{f_M} \circ \qquad FRQ_{cyto} \xrightarrow{f_d} \circ$$

$$FRQ_{cyto} \xrightarrow{0.5} FRQ_{nucleus} \qquad \mathrm{FRQ}_{nucleus} \xrightarrow{0.6} FRQ_{cyto}$$

where the *nucleus* and *cyto* subscripts identify the elements in the nucleus and the cytosol, respectively.

The model is based on the negative feedback exerted by the protein FRQ on the transcription of the frq gene; the rate of gene expression is enhanced by light. The FRQ protein is represented by FRQ_{cyto} in its cytosolic form and $\mathrm{FRQ}_{nucleus}$ in its nuclear form.

The model includes gene transcription in the nucleus, accumulation of the corresponding mRNA in the cytosol with the associated protein synthesis, protein transport into and out of the nucleus, and regulation of gene expression by the nuclear form of the FRQ protein. The function $f_{FRQ}(t)$ denoting the rate of frq transcription is defined by

[2] To better focus on the proposed methodology the example is expressed in a basic subset of the CWC, in which reactions are denoted in a standard self-explanatory way.

$$f_{FRQ}(t) = v_s(t)\frac{K_I^n}{FRQ_{nucleus} + K_I^n}$$

where

$$v_s(t) = \begin{cases} 200 \text{ when } 2iT \leq t < (2i+1)T \\ 160 \text{ when } (2i+1)T \leq t < (2i+2)T \end{cases} \qquad (i \geq 0)$$

and T represents the period of the dark-light phases.

The parameter $v_s(t)$ defined by increases in light conditions of the current time of the simulation, where T represents the period of the dark-light phases. The constant K_I is related to the threshold beyond which nuclear FRQ represses frq transcription; the Hill coefficient n, characterises the degree of cooperatively of the repression process. In the functions, the name of an atom indicates its multiplicity. The mRNA degradation is given by the Michaelis rate function

$$f_M = v_m\frac{M_{cyto}}{K_M + M_{cyto}}$$

The FRQ degradation is given by the Michaelis rate function

$$f_d = v_d\frac{FRQ_{cyto}}{K_d + FRQ_{cyto}}$$

where v_s is the maximum rate of FRQ degradation and the Michaelis constant related to this process is K_d.

[Figure 4]

[Figure 5]

As in [61] we modelled the oscillations under two different conditions: i) constant dark condition, ii) alternate light and dark phases. Following [61], the values of the parameters are set as: $v_m = 50.5$, $v_d = 140$, $k_s = 0.5$, $k_1 = 0.5$, $k_2 = 0.6$, $K_m = 50$, $K_I = 100$, $K_d = 13$, $n = 4$. Concentrations have been made discrete by scaling 1nM to 100 atomic elements. In the constant dark condition, parameter v_s is equal to 160, in the alternate condition; v_s is equal to 160 during the dark phase and to 200 during the light phase. Figure 4 shows an extract of a single stochastic simulation of the circadian oscillations in the dark/light alternate condition, plotting the number of *FRQ* proteins within the nucleus ($FRQ_{nucleus}$), the total number of *FRQ* proteins in the cell and the number of mRNA molecules leading the synthesis of *FRQ*. Figure 5 shows the outcome of the peak detection tool, which is able to summarise the frequency of the peak events over time. The plot results after capturing the peaks in the curve of the cytosolic mRNA for the FRQ protein synthesis. Measuring the distance between two consecutive peaks, we compute the period of each oscillation and then plot the moving average, over 200 simulations, of the local periods. In the constant dark condition, the circadian period is close to 21 and half hours, but increases; producing damping oscillations with a period of approximately 24 hours, in the dark/light alternate condition.

5.2 Performance evaluation

[Figure 6]

[Figure 7]

[Figure 8]

The performance of the simulator was tested on the Neurospora model, described in Sec. 5.1. We ran two set of experiments: the first one considering 8 virtual machines (VMs) each having 4 cores Intel E-2670 2.6 GHz with 20MB of L3 cache running in the Amazon Elastic Compute Cloud (Amazon EC2) [62]; the second set considering an heterogeneous environment of virtual and physical machines which allowed to scale the core count up to 96. The heterogeneous environment, which can be considered a private cloud including a public cloud comprises: 8 EC2 virtual machines with 4 virtualised cores, two workstations at University of Pisa each having 16 cores Intel Sandy Bridge @2GH.z with 20MB of L3 shared cache, and 1 workstation, at University of Torino, having 32 cores Intel Nehalem @2.0GHz with 18MB of L3 shared cache. Virtual and physical machines run Linux x86_64.

In the first test we measured the speedup and the execution time of the simulator when running 96 days of simulation time on a single quad-core VM. The results obtained are shown in Figure 6. In this case, the maximum speedup using all available cores is 3.15 out of 4 so that the execution time decreases from about 224 minutes of the sequential run down to about 71 minutes. Observe that in this case the speedup is not ideal because of the additional work needed for on-line alignment of trajectories at the simulation time. In this regard, it is worth to recap that simulation time advances along random walks, thus different simulations instances proceeds at different speed with respect to simulation time. In traditional approaches this cost is typically paid during post-processing phase.

Next, we executed the same test using 8 quad-core VMs. Figure 7 reports the speedup for the same simulation time varying the number of virtual cores used. The trend is almost ideal. With 32 virtual cores we obtained a completion time of 10.5 minutes, with a gain of about 21x with respect to the sequential execution time of the simulator on a single-core VM of the same clock frequency and a gain of ~7x with respect to the execution time obtained on the single quad-core VM.

In the second set of experiments, we executed the simulation using different platforms. Initially we ran the simulator on the 32 cores Nehalem workstation using the shared memory implementation of the simulator. The minimum execution time obtained on that machine using all cores available is 67.3 minutes (i.e. almost the same time) obtained using the 8 Amazon VMs (having an overall number of 32 virtualised cores). This result confirms the quality of the distributed implementation of the simulator. Next, in order to further decrease the simulation time, we used together the Nehalem workstation, the Amazon VMs, and the two 16 cores Sandy Bridge workstations. In this case, since the machines were not homogeneous in terms of number of cores and computational power, we used a weighted dispatching policy for the distribution of the simulations, where the weights used are the number of virtualised or physical cores of the target platform. The results obtained for the execution time and the speedup (the speedup is computed w.r.t. the execution time obtained on single-core Amazon VM), are shown in Figure 8. For this test, the analysis pipeline was mapped on the 32 cores Nehalem workstation. The minimum execution time obtained using 96 cores (32 cores in the 8 quad-core VMs, 32 cores in the Nehalem workstation and 2x16 cores in the 2 Sandy Bridge workstations) is 69.3s carrying a gain of ~62x in the execution time, which a remarkable result considering the low computation granularity (~20 ms) of the single worker thread and the high frequency of communication (30 – 80 ms) for collecting results computed by remote machines running the simulation pipeline. As a general rule, the lower the communication/computation ratio, the higher the speedup obtained. The test considered, has a not optimal communication/computation ratio and for this reason we were not able to obtain a performance improvement with more than 64 cores.

6 Conclusions

We presented the design and the implementation of the CWC simulator for the cloud, which is obtained with low engineering and coding efforts from the previous multi-core version [7, 9]. Since the CWC simulator implements a Monte Carlo algorithm for systems biology, the issues for its portable and efficient design for the cloud are paradigmatic for a broad class of algorithms for Bioinformatics. We believe that its design and implementation are also paradigmatic for the implementation of other Monte Carlo algorithms. Experimentation on both physical and virtualised execution environments (such as Amazon EC2 cloud) demonstrate that its high-level design via the FastFlow framework provides the application designer with easy engineering, seamless portability on distributed and multi-core platforms physical or virtualised, and automatic load balancing. The possibility to execute the whole simulation-analysis pipeline in the cloud make it possible to greatly reduce the data transfer from user desktop to the cloud and to deploy the tool in the cloud in a SaaS fashion. The stream-oriented design of the simulation-analysis pipeline make it possible to perform the statistical analysis and data mining of simulation results as an on-line process starting together with simulation e immediately starting to provide the user with a stream of final results, thus enforcing a fast feedback to the bioinformatics scientists. Experimental evaluations show that the design is flexible and robust with respect to target platform, and it is able to provide performance scalability also for fine-grained problems.

A recent extension of FastFlow framework supporting many-core GPGPUs via OpenCL [63], will make it possible to transparently target clusters of GPGPUs [64]. We believe that the design has the potentiality to survive in the hostile environment populated by platform heterogeneity, coding complexity and the need of performance portability.

7 Acknowledgments

This work has been partially supported by the EC-FP7 STREP project “Paraphrase” (n. 288570), the Fondazione San Paolo IMPACT project (id. ORTO11TPXK), and the BioBITs project (Converging technologies 2007, Biotechnology-ICT) - Regione Piemonte – Italy.

References

[1] D. Gillespie, “Exact stochastic simulation of coupled chemical reactions,” *J. Phys. Chem.*, vol. 81, pp. 2340–2361, 1977.

[2] R. Alur, C. Belta, and F. Ivancic, “Hybrid modeling and simulation of biomolecular networks,” in *Proc. of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC), Rome, Italy*, ser. Lecture Notes in Computer Science, vol. 2034. Springer, 2001, pp. 19–32.

[3] L. Milanesi, P. Romano, G. Castellani et al., “Trends in modeling biomedical complex systems”, *BMC Bioinformatics,* vol. 10, 2009.

[4] A. Rambaut and N. Grass, “Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees” *Comput Appl Biosci*, vol. 13, 1997, pp. 235–238.

[5] F. Da Silva and H. Senger, “Improving scalability of Bag-of-Tasks applications running on master–slave platforms”, *Parallel Computing*, vol. 35, 2009, pp. 57–71.

[6] L. Hong and L. Petzold, “Efficient parallelization of the stochastic simulation algorithm for chemically reacting systems on the graphics processing unit,” *International Journal of High Performance Computing Applications*, vol. 24, no. 2, pp. 107–116, 2010.

[7] M. Aldinucci, M. Coppo, F. Damiani et al., “On parallelizing on-line statistics for stochastic biological simulations,” in *Euro-Par 2011 Workshops, Proc. of the 2nd Workshop on High Performance Bioinformatics and Biomedicine (HiBB)*, ser. LNCS, vol. 7155. Springer, 2012, pp. 3–12.

[8] M. Aldinucci, A. Bracciali, P. Liò et al., “StochKit-FF: Efficient systems biology on multicore

architectures," in *Euro-Par 2010 Workshops, Proc. of the 1st Workshop on High Performance Bioinformatics and Biomedicine (HiBB)*, ser. LNCS, vol. 6586. Springer, 2011, pp. 167–175.

[9] M. Aldinucci, M. Coppo, F. Damiani et al., "On designing multicore-aware simulators for biological systems," in *Proc. of Intl. Euromicro PDP 2011: Parallel Distributed and network-based Processing*. Ayia Napa, Cyprus: IEEE, 2011, pp. 318–325.

[10] M. Coppo, F. Damiani, M. Drocco et al., "Simulation Techniques for the Calculus of Wrapped Compartments," *Theoretical Computer Science*, vol. 431, pp. 75–95, 2012.

[11] M. Aldinucci, M. Danelutto, P. Kilpatrick et al., "Fastflow: high-level and efficient streaming on multi-core," in *Programming Multi-core and Many-core Computing Systems*, ser. Parallel and Distributed Computing. Wiley, 2013, ch. 13.

[12] M. Aldinucci, S. Campa, M. Danelutto et al., "Targeting distributed systems in fastflow," in *Euro-Par 2012 Workshops, Proc. of the CoreGrid Workshop on Grids, Clouds and P2P Computing*, ser. LNCS, vol. 7640. Springer, 2013, pp. 47–56.

[13] A. Ferscha, "Performance Models for Discrete Event Systems with Synchronisations: Formalisms and Analysis Techniques," ser. MATCH Advanced Schools, Jaca, Spain, Sep. 1998, vol. 2, ch. VII – Simulation.

[14] I. Park, M. J. Voss, S. W. Kim et al., "Parallel programming environment for OpenMP," *Scientific Programming*, vol. 9, pp. 143–161, 2001.

[15] P. S. Pacheco, *Parallel programming with MPI*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[16] M. Cole, "Bringing skeletons out of the closet: A pragmatic manifesto for skeletal parallel programming," *Parallel Computing*, vol. 30, no. 3, pp. 389–406, 2004.

[17] K. Asanovic, R. Bodik, J. Demmel et al., "A view of the parallel computing landscape," *Communications of the ACM*, vol. 52, no. 10, pp. 56–67, 2009.

[18] H. González-Vélez and M. Leyton, "A survey of algorithmic skeleton frameworks: High-level structured parallel programming enablers," *Software: Practice and Experience*, vol. 40, no. 12, pp. 1135–1160, 2010.

[19] M. Vanneschi, "The programming model of ASSIST, an environment for parallel and distributed portable applications," *Parallel Computing*, vol. 28, no. 12, pp. 1709–1732, Dec. 2002.

[20] W. Thies, M. Karczmarek, and S. P. Amarasinghe, "StreamIt: A language for streaming applications," in *Proc. of the 11th Intl. Conference on Compiler Construction (CC)*. Springer, 2002, pp. 179–196.

[21] I. Buck, T. Foley, D. Horn et al., "Brook for GPUs: stream computing on graphics hardware," in *ACM SIGGRAPH '04 Papers*. New York, NY, USA: ACM Press, 2004, pp. 777–786.

[22] Intel Corp., *Threading Building Blocks*. [Online] http://www.threadingbuildingblocks.org/

[23] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In Usenix OSDI '04, 2004, pp. 137–150.

[24] V. González-Vélez and H. González-Vélez, "Parallel Stochastic Simulation of Macroscopic calcium currents," *J. Bioinformatics and Computational Biology*, vol. 5, no. 3, 2007, pp. 755–772.

[25] T. Cickovski, K. Aras, M. Swat et al. "From Genes to Organisms Via the Cell: A Problem-

Solving Environment for Multicellular Development," *Computing in Science and Eng*., vol. 9, no. 4, 2007, pp. 50–60.

[26] L. D. Stein, "The case for cloud computing in genome informatics," *Genome Biol*, vol. 11, no. 5, p. 207, 2010.

[27] M. Schatz, B. Langmead, and S. Salzberg, "Cloud computing and the DNA data race," *Nature biotechnology*, vol. 28, no. 7, p. 691, 2010.

[28] H. Li and N. Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics*, vol. 11, no. 5, 2010, pp. 473–483.

[29] B. Langmead, M. Schatz, J. Lin et al., "Searching for snps with cloud computing," Genome Biol, vol. 10, no. 11, p. R134, 2009.

[30] Q. Zou, X.B. Li, W.R. Jiang et al., Survey of MapReduce frame operation in bioinformatics. *Briefings in Bioinformatics*, 2013. In press. DOI: 10.1093/bib/bbs088

[31] *Apache Software Foundation.* Hadoop, 2013 (last accessed). [Online] http://hadoop.apache.org

[32] M. Schatz, "CloudBurst: highly sensitive read mapping with MapReduce," *Bioinformatics*, vol. 25, no. 11, pp. 1363–1369, 2009.

[33] B. Langmead, K. Hansen, and J. Leek, "Cloud-scale RNA-sequencing differential expression analysis with myrna," *Genome Biol*, vol. 11, no. 8, p. 83, 2010.

[34] G. Ropella and C. Hunt, "Cloud computing and validation of expandable in silico livers," *BMC systems biology*, vol. 4, no. 1, p. 168, 2010.

[35] T. Eissing, L. Kuepfer, C. Becker et al., "A computational systems biology software platform for multiscale modeling and simulation: integrating whole-body physiology, disease biology, and molecular reaction networks," *Frontiers in Physiology*, vol. 2, 2011.

[36] *Folding@home*. [Online] http://folding.stanford.edu/English/HomePage

[37] Y. Chen, C. Lawless, C.S. Gillespie et al., "CaliBayes and BASIS: integrated tools for the calibration, simulation and storage of biological simulation models. *Briefings in Bioinformatics*, vol. 11, no. 3, 2010, pp. 278–89.

[38] M. Sevior, T. Fifield, and N. Katayama, "Belle Monte-Carlo production on the Amazon EC2 cloud," in *Journal of Physics: Conference Series*, vol. 219, no. 1. IOP Publishing, 2010.

[39] B. Drawert, S. Engblom, A. Hellander, "URDME: A modular framework for stochastic simulation of reaction-transport processes in complex geometries", *BMC Systems Biology,* vol. 6, 2012.

[40] StochSS: Stochastic Simulation Service A Cloud Computing Framework for Modeling and Simulation of Stochastic Biochemical Systems. [Online] http://iguana.cs.ucsb.edu/wordpress/

[41] H. Miras, R. Jimenez, C. Minas et al. "CloudMC: a cloud computing application for Monte Carlo Simulation", Phys Med Biol, vol. 58, pp 125-133, 2013.

[42] J. Shendure and H. Ji. "Next-generation DNA sequencing", *Nature biotechnology*, vol. 26, no. 10, 2008, pp. 1135–1145.

[43] A. Matsunaga, M. Tsugawa, and J. Fortes, "Cloudblast: Combining MapReduce and virtualization on distributed resources for bioinformatics applications", in eScience, 2008. In *Proc. of the 4th IEEE Intl. Conference on eScience*, IEEE, 2008, pp. 222–229.

[44] A. Phillips and L. Cardelli, “Efficient, correct simulation of biological processes in the stochastic pi-calculus,” in *In Proc. of Intl. Conference on Computational Methods in Systems Biology (CMSB), Edinburgh, Scotland*, ser. LNCS, vol. 4695. Springer, 2007, pp. 184–199.

[45] S. Ramsey, D. Orrell, and H. Bolouri, “Dizzy: Stochastic simulation of large-scale genetic regulatory networks (supplementary material),” *J. Bioinformatics and Computational Biology*, vol. 3, no. 2, pp. 437–454, 2005.

[46] F. Ciocchetta and J. Hillston, “Bio-PEPA: An extension of the process algebra PEPA for biochemical networks,” in *Proc. of 1st Workshop "From Biology To Concurrency and back (FBTC), Lisbon, Portugal*, ser. ENTCS, vol. 194, no. 3. Elsevier, 2008, pp. 103–117.

[47] T. Ray and P. Saini, “Engineering design optimization using a swarm with an intelligent information sharing among individuals,” *Eng. Opt.*, vol. 33, pp. 735–748, 2001.

[48] P. K. Dhar, T. C. Meng, S. Somani et al., “Grid cellware: the first grid-enabled tool for modelling and simulating cellular processes,” *Bioinformatics*, vol. 7, pp. 1284–1287, 2005.

[49] J. Barnat, L. Brim, and D. Safránek, “High-performance analysis of biological systems dynamics with the divine model checker,” *Briefings in Bioinformatics*, vol. 11, no. 3, pp. 301–312, 2010.

[50] L. Petzold, *StochKit: stochastic simulation kit web page*, [Online] http://www.engineering.ucsb.edu/~cse/StochKit/index.html

[51] J. Intosalmi, T. Manninen, K. Ruohonen et al., “Computational study of noise in a large signal transduction network,” *BMC bioinformatics*, vol. 12, no. 252, 2011.

[52] H. Salis, V. Sotiropoulos, and Y. Kaznessis, “Multiscale hy3s: hybrid stochastic simulation for supercomputers,” *BMC bioinformatics*, vol. 7, no. 93, 2006.

[53] G. Gruenert, B. Ibrahim, T. Lenser et al., “Rule-based spatial modeling with diffusing, geometrically constrained molecules,” *BMC bioinformatics*, vol. 11, no. 307, 2010.

[54] G. Klingbeil, R. Erban, M. Giles et al., “StochSimGPU: parallel stochastic simulation for the systems biology toolbox 2 for MATLAB,” *Bioinformatics*, vol. 27, no. 8, p. 1170, 2011.

[55] *FastFlow website*. [Online] http://mc-fastflow.sourceforge.net

[56] G. Kahn, “The semantics of a simple language for parallel programming,” Proc. of *Information processing*, Stockholm, Sweden, 1974. North Holland, pp. 471–475.

[57] M. Aldinucci, M. Danelutto, P. Kilpatrick et al., “An efficient unbounded lock-free queue for multi-core systems,” in *Proc. of 18th Intl. Euro-Par 2012 Parallel Processing*, ser. LNCS, vol. 7484. Springer, 2012, pp. 662–673.

[58] *ZeroMQ website*, 2012. [Online] http://www.zeromq.org/

[59] M. Coppo, F. Damiani, M. Drocco et al., “Stochastic Calculus of Wrapped Compartments,” in *Proc. of 8th Workshop on Quantitative Aspects of Programming Languages (QAPL)*, vol. 28. ser. EPTCS, vol. 28, 2010, pp. 82–98.

[60] *CWC Simulator project*, Sourceforge website, 2013 (last accessed). [Online] http://sourceforge.net/projects/cwcsimulator/

[61] J. Leloup, D. Gonze, and A. Goldbeter, “Limit cycle models for circadian rhythms based on transcriptional regulation in drosophila and Neurospora,” *Journal of Biological Rhythms*, vol. 14, no. 6,

1999.

[62] *Amazon Elastic Compute Cloud*. [Online] http://aws.amazon.com/ec2/

[63] *OpenCL*, Khronos Compute Working Group. [Online] http://www.khronos.org/opencl/

[64] M. Goli, M. Garba, and H. González-Vélez, "Streaming dynamic coarse-grained CPU/GPU workloads with heterogeneous pipelines in FastFlow," in *Proc. of the 14th IEEE Intl. Conference on High Performance Computing and Communications (HPCC)*. IEEE, 2012, pp. 445–452.

[65] C. Blilie, "Patterns in Scientific Software: an Introduction," *Computing in Science and* Engineering, vol 4, no. 3, 2002, pp. 48–53.

List of figures and tables

Figure 8: Performance of the simulator for the Neurospora model on an heterogeneous environment of virtualised and physical machines (8 quad-core Amazon EC2 VMs, 1 32-core Nehalem workstation, and 2 16-core Sandy Bridge workstations): speedup and execution time varying the number of cores.

Authors' biography

Marco Aldinucci is an assistant professor at Computer Science Department of the University of Torino since 2008. Previously, he has been researcher at University of Pisa and Italian National Research Agency. He is the author of over a hundred papers in international journals and conference proceeding. He has been participating in over 20 research projects concerning parallel computing. He is the recipient of the HPC Advisory Council University Award 2011, and NVidia CUDA Research center award 2013. He has been leading the "Low-Level Virtualization and Platform-Specific Deployment" workpackage within the EU-STREP FP7 ParaPhrase project. His research is focused on parallel and distributed computing.

Massimo Torquati is a researcher at the Computer Science Department of the University of Pisa. He has more than 15 years experience in software design, he participated to the design and the implementation of several compilers and frameworks for parallel programming both in academic and industrial settings, inter-alia SkIE, ASSIST, VirtuaLinux, FastFlow. He co-authored over 30 papers appeared in proceedings of international conferences and journals. His main research area is language and algorithms for parallel computing.

Concetto Spampinato received the PhD in Computer Engineering from University of Catania in 2008 where he is Research Assistant. He has worked actively on object detection, tracking, behaviour understanding and even detection in complex and noisy environments. He has been working to EU-STREP FP7 Fish4Knowledge project and the AQUACAM research program. He also worked on machine learning atlas-guided approaches for 2D medical image segmentation and knowledge discovery in biomedicine. He has published over 90 articles in international journals and refereed conference proceedings.

Maurizio Drocco is a master student and software engineer at the university of Torino. He has been participating to the FastFlow project and BioBITs project projects. He is the main developer of the CWC simulator. He has co-authored over 10 papers articles in international journals and refereed conference proceedings.

Claudia Misale received her MSc degree cum laude in Computer Science at University of Calabria in 2012 and is currently a PhD student at Computer Science Deparment of the

University of Torino, Italy. She has been participating to the FastFlow project. Her research is focused on high-performance tools for bioinformatics.

Cristina Calcagno received her MSc degree cum laude in Plant Biology at University of Torino in 2007, and the PhD in Science and High Technology - Biology and Biotechnology of fungi at the same university. From February 2011 to July 2012 she has been postdoctoral researcher at the Department of Computer Science within the project BioBITs project. Her main research interests are: molecular biology, plant biology, systems biology, nutrigenetic and nutrigenomic.

Mario Coppo is a full professor of Computer Science at the Computer Science Department of the University of Torino since 1987. He authored about 70 papers appearing in journals and international conference proceedings. He has leading several research projects funded by National and European institutions and has been the Director of the Computer Science Department from 2004 to 2010. His main research topics are concurrency theory and programming language semantics.

System	Description	Used Resources	Cloud	How to use	Ref
CloudBurst	System to map sequence data to a reference genome	Apache Hadoop	Amazon EC2 Homepage	Source code to be compiled and executed on a Hadoop cluster	[37]
Computational Systems Biology Software Suite Bayer	Platform for computational biology by integrating body physiology, disease biology, and molecular reaction networks	Apache Hadoop PK-Sim, MoBi R and Matlab	D-Grid GmbH Homepage	Executable to be installed	[40]
Crossbow	Software pipeline for genome resequencing analysis	Bowtie SoapSNP	Amazon EC2 Homepage	Crossbow Web Application	[36]
Folding@home	Protein folding simulation	--	Windows Azure Homepage	Folding@home website	[41]
Myrna	Calculate differential gene expression in RNA-seq datasets	Bowtie R/Bioconductor Apache Hadoop	Amazon EC2 Homepage	Myrna Web Application	[38]

Table 1: Comparison of some cloud based computational biology and bioinformatics tools available on the Web.

Tool	Calculus	Simulation Schema	Parallelism	Data Analysis	Ref
SCWC	CWC	Gillespie	FastFlow	online statistics	[31]
SPiM	π-calculus	Gillespie	none	none	[19]
Dizzy	Reaction Model	Gillespie, Tau-Leap, ODE	none	none	[20]
BioPEPA	Process Algebra	ODE, Gillespie	none	none	[21]
Cellware	Reaction Model	Gillespie, Gibson-Bruck, ODE	none	none	[23]
DiVinE	Model Checker	ODE	MPI	none	[24]
StochKit	Reaction Model	Gillespie, Tau-leaping	MPI	post-processing	[25]
StochKit2	Reaction Model	Gillespie, Tau-leaping	POSIX threads	post-processing	[25]
StochKit-FF	Reaction Model	Gillespie, Tau-leaping	FastFlow	online statistics	[5]
Hy3S	Reaction Model	Gibson-Bruck, Hybrid	MPI	post-processing	[27]
StochSimGPU	Reaction Model	Gillespie, Gibson-Bruck, Li	NVidia CUDA	post-processing	[29]

Table 2: Schematic comparison of the main features of several biological simulation tools.

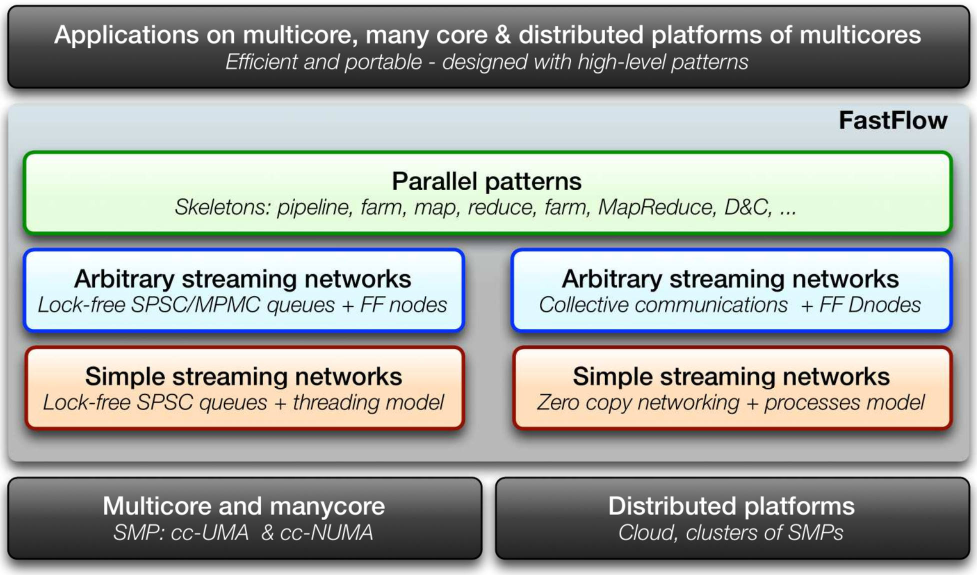
Applications on multicore, many core & distributed platforms of multicores
Efficient and portable - designed with high-level patterns
FastFlow
Parallel patterns
Skeletons: pipeline, farm, map, reduce, farm, MapReduce, D&C, ...
Arbitrary streaming networks
Lock-free SPSC/MPMC queues + FF nodes
Arbitrary streaming networks
Collective communications + FF Dnodes
Simple streaming networks
Lock-free SPSC queues + threading model
Simple streaming networks
Zero copy networking + processes model
Multicore and manycore
SMP: cc-UMA & cc-NUMA
Distributed platforms
Cloud, clusters of SMPs

Figure 1: Layered architecture of the FastFlow programming framework.
195x115mm (300 x 300 DPI)

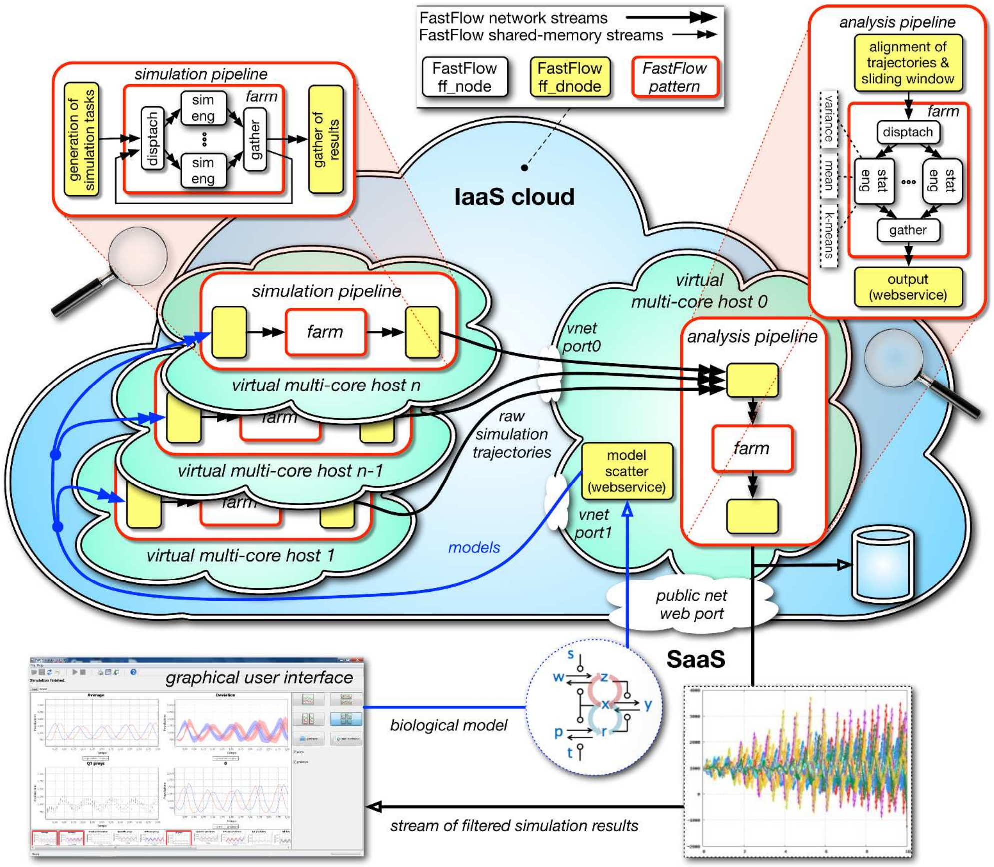

Figure 2: Architecture of the CWC parallel simulator with on-line parallel analysis. The FastFlow framework automatically generates the implementation of patterns connecting ff_nodes and ff_dnodes with streams, which are implemented either in the shared-memory model within the single virtual machine or in the message-passing model across virtual machines.
352x308mm (300 x 300 DPI)

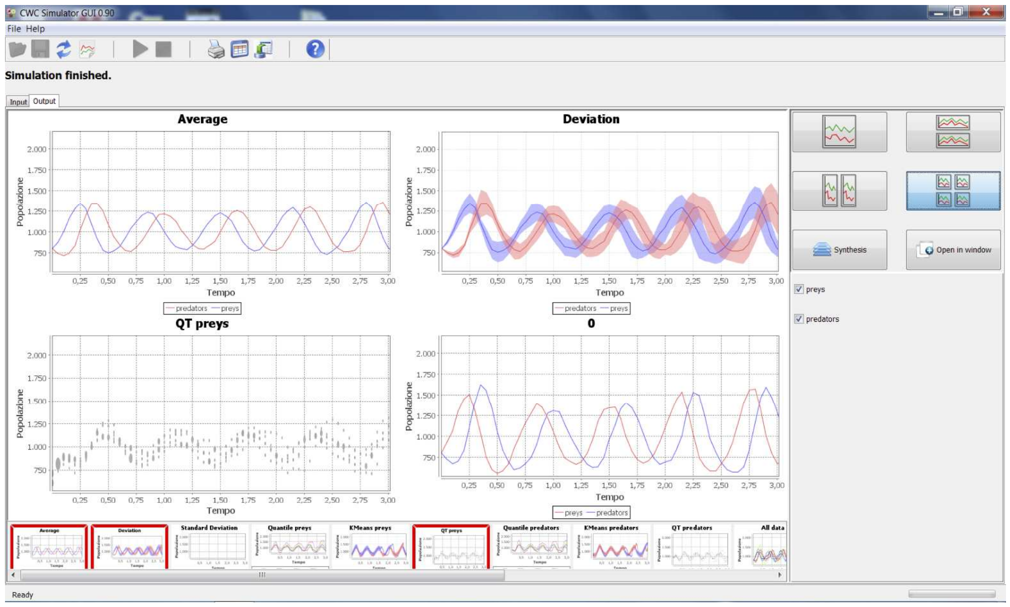

Figure 3: Screenshot of the simulation tool interface: output analysis plotting. The interface enables the usage of the application in a SaaS fashion.
592x354mm (150 x 150 DPI)

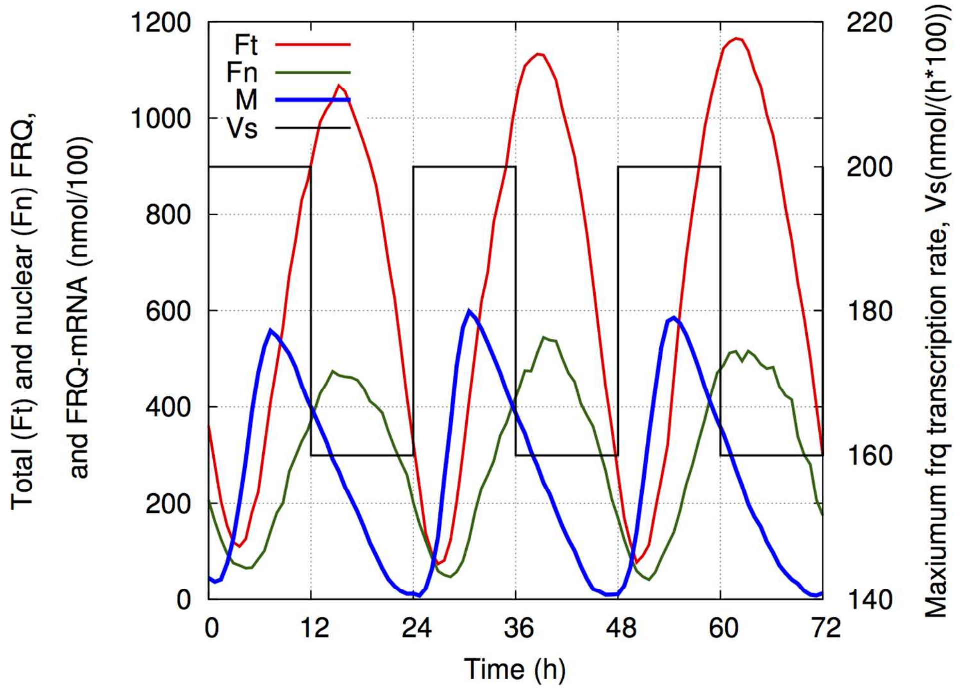

Figure 4: An extract (72 hours of simulated time) of a single stochastic simulation of the circadian oscillations in the dark/light alternate condition (Vs), plotting the number of FRQ proteins within the nucleus (Fn), the total number of FRQ proteins in the cell (Ft) and the number of mRNA molecules leading the synthesis of FRQ (M).
127x88mm (300 x 300 DPI)

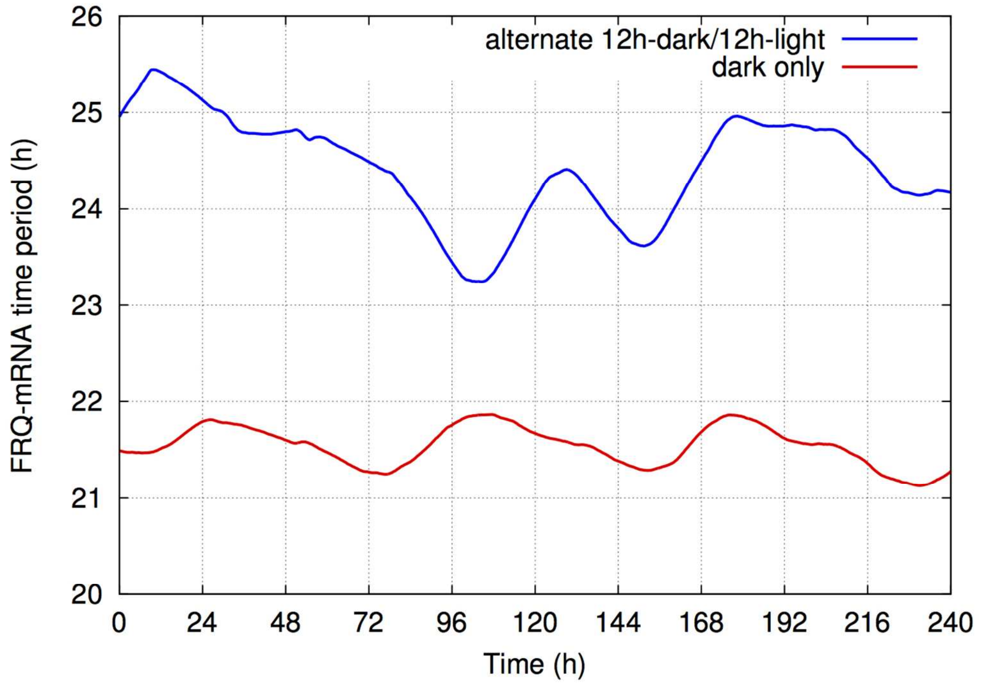

Figure 5: Output of the simulation-analysis workflow with peak detection analysis module applied to FRQ(M) for both alternate dark/light and dark conditions. The frequency of the peak events over time is shown along evolution of 10 days of simulated time.
127x88mm (300 x 300 DPI)

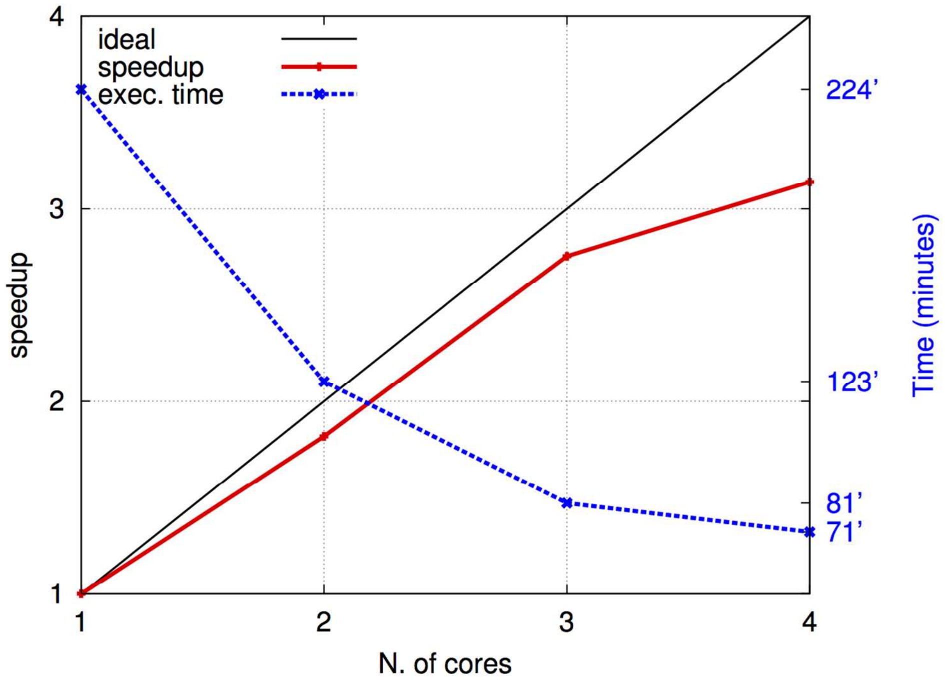

Figure 6: Performance of the simulator for the Neurospora model in a single quad-core VM in the Amazon EC2 cloud: speedup and execution time varying the number of virtualised cores.
127x88mm (300 x 300 DPI)

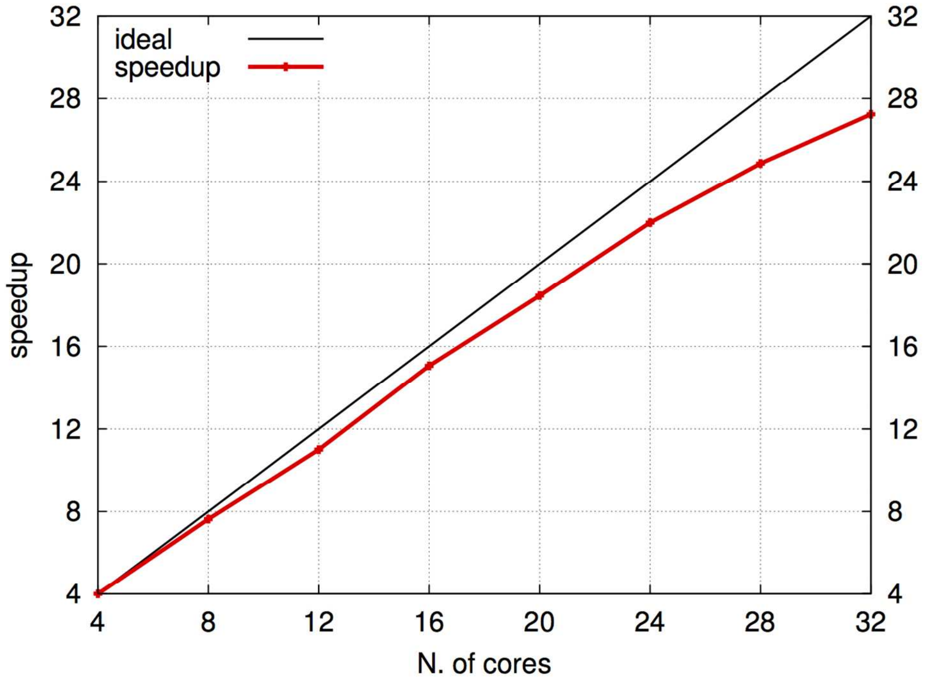

Figure 7: Performance of the simulator for the Neurospora model on a virtual cluster of 8 quad-core VMs in the Amazon EC2 cloud: speedup varying the number of virtualised cores.
127x88mm (300 x 300 DPI)

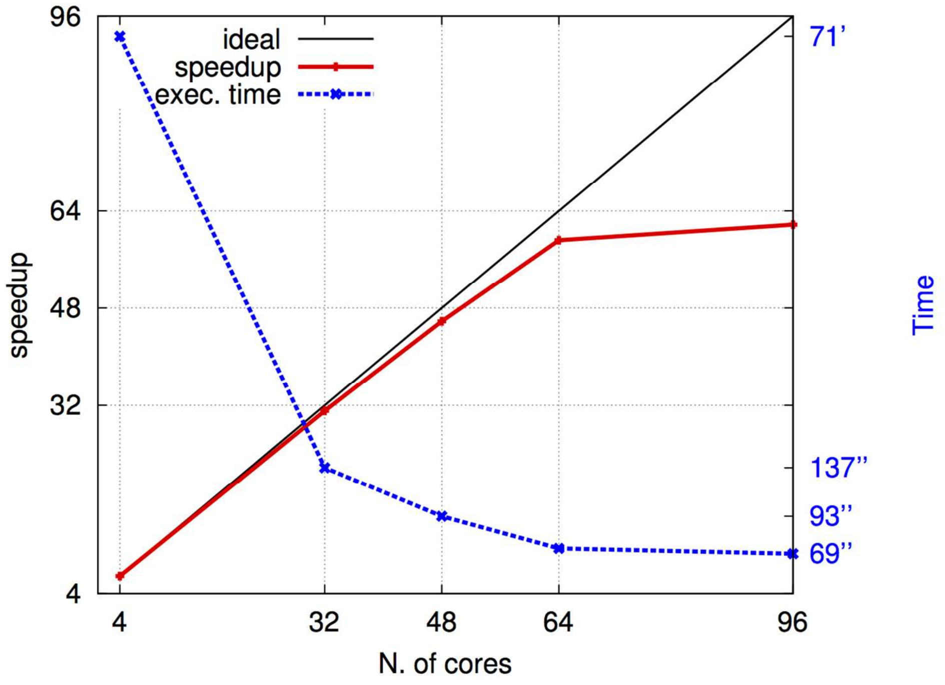

Figure 8: Performance of the simulator for the Neurospora model on an heterogeneous environment of virtualised and physical machines (8 quad-core Amazon EC2 VMs, 1 32-core Nehalem workstation, and 2 16-core Sandy Bridge workstations): speedup and execution time varying the number of cores.
127x88mm (300 x 300 DPI)