SPECIAL ISSUE PAPER

# Planting parallel program simulation on the cloud

Antonio Cuomo [1],[*],[†], Massimiliano Rak [2] and Umberto Villano [1]

[1]*Department of Engineering, University of Sannio, Piazza Roma 21, 82100 Benevento, Italy*
[2]*Department of Information Engineering, Second University of Naples, Via Roma 29, 81031, Aversa, Italy*

## SUMMARY

The writing of efficient parallel code has always been a tedious and time-consuming process. However, the performance prediction prototype tools devised in the last decade come of age now, thanks to the availability of the almost unlimited computing power of clouds. This paper presents the practical use of mJADES, a novel environment for running multiple concurrent simulations in the cloud, to predict the performance of parallel code in multiple working conditions at once. After an introduction on mJADES and its operational aspects, the construction of parallel code performance prediction models will be dealt with. The models and the results obtained for a simple but complete and meaningful case study will be presented, discussing the accuracy obtained by simulation and the time required for performing the whole set of simulations necessary to characterize the program behavior. Copyright © 2013 John Wiley & Sons, Ltd.

## 1. INTRODUCTION

Parallel programming has always been a difficult matter. Even now that it is widely practiced, due to the wide diffusion of multi-core CPUs, writing performance-efficient code remains a tedious and time-consuming process. As a matter of fact, the design and tuning of parallel code, whether for shared-memory or distributed-memory architectures, is a challenging problem, and one for which no general rules and techniques exist.

For the aforementioned reasons, in the last decade, many researchers have suggested the use of performance-driven software engineering techniques. [1–5] Just to give the reader the flavor of the proposal, the idea is to adopt performance prediction techniques as early as in the software design step to avoid using program structures and patterns that would inevitably lead to low performance. The tools used for program performance prediction were almost always based on simulation (at different levels of granularity, depending of the particular tool). The drawback of the performance-driven approach is essentially the time required for running reasonably accurate simulations. Developers do not like to wait. They prefer to write inefficient programs, to run them, to discover that they are far away from the desired performance, and to start all over again, possibly asking for help from a skilled developer. But now things have changed. The almost unlimited, low-cost computing power that can be leased from a cloud makes it possible to run multiple lengthy simulations at once. In our opinion, this is a wonderful opportunity to look back at the old performance prediction tools and to 'plant' them on the cloud in the hope that their utility could be finally widely recognized.

*Correspondence to: Antonio Cuomo, Department of Engineering, University of Sannio, Piazza Roma 21, 82100 Benevento, Italy.

[†]E-mail: antonio.cuomo@unisannio.it

Parallel program performance prediction aims at gaining insight on a parallel code performance behavior without resorting to extensive performance testing. The idea is to know with reasonable accuracy how the code will perform without executing it many times on the real hardware and, possibly, even in the absence of a fully developed code. Our research group has shown that this is possible through the simulation of the program behavior. [6, 7] Most of our past results have been obtained by running a program simulator fed with actual or synthetic traces. [8–11] Though decidedly faster than actual program executions, simulation of a complex code running on a high number of processors is a time-consuming process. The construction of performance diagrams of a given code (most of the times, the evaluation of speedup for progressively higher number of processors until the knee of the curve is reached) requires many simulations. This is exactly the point where *concurrent simulation* comes into play. We will show that, given a parametric simulation model, the required simulations can be straightly executed in a single step through the mJADES system.

The mJADES is a discrete-event simulation environment [12] running in the cloud in software-as-a-service mode. In practice, it is a cloud application that allows its users to execute multiple simulations leasing resources from the cloud. These simulations are executed *concurrently*, in that the inherent parallelism of the cloud is exploited to execute multiple sequential simulation tasks at the same time. This is useful whenever multiple simulations are required, for example, to sweep one or several simulation parameters, or to answer to 'what if' system configuration questions. This is not the canonical approach; as in the past, almost always, parallel hardware has been used to speed up a single simulation, leading to the development of parallel and distributed simulation techniques. [13–15]

The mJADES aims at running sequential, monolithic discrete-event simulations, with the possibility to run multiple simulation instances at the same time, leasing automatically a suitable number of computing resources from a cloud. As discussed in the paper introducing mJADES, [12] the rationale of this design is clear. Whenever a simulation must be executed many times, there is an exploitable source of parallelism in running together multiple instances of the same basic simulation task. This happens when simulations are stochastic and should be repeated to perform statistical validation or for parameter-sweep simulations, where one or multiple input parameters vary over a range of values. The latter is exactly the case with parallel code performance prediction, where it is necessary to repeat simulation for many different values of number of processors or to sweep a range of values of a decomposition parameter to determine the optimal setting. In this paper, we will present our experiences of the use of mJADES for the performance prediction of a parallel Message Passing Interface (MPI) program, the Jacobi iteration. The example has been carefully chosen to be simple but complete and meaningful. After a discussion on the modeling phase, we will show the accuracy obtained by simulation (by comparing the predicted results to the actually-measured ones) and the time required for simulation.

The remainder of this paper is structured as follows: the next section presents related work, pointing out the original features of our approach. Section 3 gives an overview of mJADES from an operative point of view, with references to companion papers describing its architectural features. Section 4 describes the model development process. The case study on the use of mJADES for parallel software performance prediction is dealt with in section 5. Section 6 draws the conclusions and sketches our future research directions.

## 2. BACKGROUND AND RELATED WORK

The mJADES simulation environment, introduced in [12], exploits cloud resources by running multiple concurrent instances of a traditional sequential simulator. In this section, we review the literature that deals with the interaction of simulation engines, distributed programming, and cloud systems, and present existing work on simulation-based program performance prediction.

### 2.1. Process-oriented discrete-event simulation: a Java perspective

A framework running multiple sequential simulations on cloud resources can be built on the top of any type of simulation engine. However, mJADES exploits a Java-based engine for process-

oriented discrete-event simulation (JADES) developed by our research group and presented in [16]. The aforementioned reference presents a review of Java-based simulators.

### 2.2. Distributed simulation

As discussed in the introduction, mJADES exploits cloud computing resources by running multiple concurrent instances of a traditional sequential simulator. This makes it fairly different from the majority of the systems developed in the main body of research on parallel and distributed simulation. [13–15] In these systems, the simulation model is decomposed into entities allocated on multiple processor systems, whether shared-memory (e.g., ComplexSim [17]) or distributed-memory ones (e.g., *ns-3* [18]). In parallel and distributed simulation systems, groups of entities are assigned to logical processes, which evolve with their own simulation clock. This entails the need to keep the different logical processes synchronized; a problem for which different solutions have been proposed, including conservative [19] and optimistic [20] protocols. A recent survey on Java-based parallel simulators can be found in [21].

   The mJADES relies on a completely different approach, because individual simulation runs are not decomposed among multiple processors. mJADES instead speeds up simulation jobs by launching multiple concurrent runs ('replications') at a time. This technique is not new and was explored in a seminal paper by Heidelberger in the eighties, [22] where the Authors compared the obtained performance to the one of a more conventional parallel simulator. As a matter of fact, in the last two decades, the dominance of clusters and grid systems focused the interest of researchers on parallel simulation engines. Nowadays, the cloud approach, which is based on web transactions managed by independent tasks, has made the simulation paradigm, on the basis of concurrent sequential runs, more attractive than in the past.

### 2.3. Simulation-as-a-service

The 'as-a-service' methodology originates in the world of service-oriented architecture. In the literature, there are several contributions on the 'as-a-service' provisioning of simulation software. The paper [23] presents a Java-based architecture for replicated simulations on a network of processors, called 'alliance'. Unlike this work, mJADES relies on current cloud technologies and exploits an open-source simulator engine instead of a closed-source one (Silk, [24]). Some architectural aspects linked to the simulation-as-a-service paradigm (the provision of web services interfaces for simulations and the translation of simulation requests to XML) are tackled in [25]. Reference [26] discusses issues related to the design of simulation web services and their composition into simulation workflows. In [27], SimSaaS or simulation software-as-a-service is introduced. This is a simulation framework with a focus on the management of multi-tenancy.

### 2.4. Simulation and cloud computing

Papers on the use of cloud resources for computer simulation are just beginning to appear in the literature. Most of them deal with the porting of 'traditional' parallel/distributed simulation schemes to a public or private cloud. In [28], Park and Fujimoto present Aurora, an optimistic parallel discrete-event simulator based on a master–worker approach. This structure makes Aurora more suitable than other simulators on the basis of optimistic synchronization [29] for utility computing systems. D'Angelo [30] argues that the high variability in performance that is typical of cloud environments must be taken explicitly into account. His research team is developing Artis/GAIA+, a middleware for parallel and distributed simulations that aims at balancing adaptively the simulation by model decomposition, dynamic partitioning, and bottleneck removal. In [31], the authors present an application of a master–worker platform based on volunteer computing to solve simulation models requiring many runs. Unlike in mJADES, the generation of tasks and the analysis of results is not performed automatically.

## 2.5. Open-source cloud middleware

Nowadays, cloud computing is still a hot research topic. Public providers are growing in number, and open-source middleware designed for private clouds has reached sufficient maturity to be deployed in commercial and research environments. Reference [32] is an up-to-date comparison of the most used infrastructure-as-a-service frameworks, namely Eucalyptus, Nimbus, OpenNebula, and Open-Stack. Research prototype software providing less common features include PerfCloud, [33] which aims at adding performance-oriented services to customary infrastructure-as-a-service provision, and CLEVER, [34] which aims at obtaining scalability and fault tolerance.

Recently, the attention of the cloud community is shifting to higher-level architectural layers, as platform-as-a-Service. In this field, currently dominated by proprietary solutions, open-source frameworks are beginning to spread. They will allow the development of cloud applications portable among different providers. Reference [35] discusses the emerging open-source proposals, also introducing mOSAIC, the open-source cloudware used to run mJADES. Details about the mOSAIC platform are discussed in [36–38].

## 2.6. Simulation-based performance prediction of parallel programs

The application of simulation techniques to the performance prediction of distributed-memory parallel programs dates back to the early nineties. In addition to the work by some of the authors of this paper, [6, 10] early efforts include the execution-driven MPI-Sim [39] and the trace-driven Dimemas network simulator. [40] Among the most recent contributions in the literature, Phantom [41] is a hybrid prediction framework that uses direct execution of computational blocks together with communication traces. Optimized replay techniques are used to obtain the duration of computational blocks on a small sample of the target machine, but trace collection requires the execution of the application on a host with the same number of processes and the same problem size. MPI-PERF-SIM [42] is a performance simulator of hierarchical clusters. It uses a hybrid benchmark/model-based approach with polynomial regression to predict performance of computational blocks, and linear regression to predict communication time. SST (Structural Simulation Toolkit) [43] is a modular architecture simulator with pluggable modules for component modeling. This enables the user to choose dynamically between fast and detailed simulations (e.g., using statistical versus cycle-accurate CPU models). SMPI (Simulated MPI) [44] is an online simulator, part of the SimGrid project. It provides analytical models for network contention.The underlying simulator is sequential. BSIM [45] is targeted at large-scale systems. It works by skeletonizing the source code (substituting intensive computational code with delays) and replacing MPI communication calls with simulated calls. Big-Sim [46] is an on-line simulator that models sequential code on target platforms by user-supplied estimates or by applying a scaling factor to measurements on a host machine.

## 3. MJADES OVERVIEW

As said in the introduction, the main goal of our work is to provide an infrastructure that allows parallel program developers to predict the impact of the design choices made for their applications, by performing quantitative performance comparisons. If simulation is used for predictions (in fact, it is often the only viable option), this entails running a very high number of simulation runs so as to validate statistically the results for each configuration of interest and to make it possible to compare them. This can be boring and time-consuming. To make practical the use of simulation for parallel program performance prediction at the early stages of software development, the process should be fast and automated. This can be translated in the following requirements for the simulation environment: (i) parallel program developers should be provided with tools to easily derive simulation models from their code, and to describe and to launch a large set of simulations automatically; (ii) the simulator should be able to predict parallel application performances in a short time; (iii) the simulation management system should be able to coordinate the execution of multiple, possibly of many, simulation runs at a time.

Our claim is that these requirements can be met by employing cloud technologies. In recent companion papers, [12, 47] we have presented *mJADES*, a cloud-based simulation application developed using the mOSAIC framework [36–38] and the JADES simulation library. [16] mJADES enables the execution of concurrent sequential simulations on computing resources leased from a cloud. We think that such a system is an ideal candidate for the fulfillment of the requirements mentioned earlier in the text. As for requirements, (i) the use of model annotations and of job decomposition techniques provides an easy way to generate simulation interfaces and to specify concisely the set of simulation to be executed, respectively; (ii) the underlying JADES simulation engine offers very good performance compared to other commonly used simulators; [16] (iii) the adoption of the mOSAIC framework makes it possible to execute many concurrent simulation tasks on the cloud, providing automatic scaling and load balancing. The remainder of this section will be devoted to discuss these statements.

### 3.1. mJADES in brief

A detailed description of the mJADES architecture is provided in [12], to which the interested reader is referred. Here, we will introduce only the essentials concepts and the terminology necessary to understand how the system operates. In brief, mJADES is a system for the concurrent execution of multiple sequential simulations on cloud resources, composed of three modules:

- the *Manager* module, which coordinates the distributed simulation execution.
- the *Core* module, which executes single simulations developed with the JADES library.
- the *Analyzer* module, which performs statistical analysis and produces graphical reports of the results provided by the Core module.

Each module is implemented as a *cloudlet*. Cloudlets, provided by the mOSAIC framework, are event-driven, asynchronous components, which are executed on special containers and are able to self-scale and to interact with any kind of cloud resource. Through these components, the mJADES cloud application offers simulation-as-a-service, available to the end user through an hypertext-transfer-protocol-based interface. The provided services are the following:

- **Model execution**: Allows the users to start a set of concurrent simulation tasks using one of the models available. A detailed explanation of this service is provided in subsection 3.2.
- **Model submission**: Allows the users to submit a new simulation model. This service is described in subsection 3.3.

### 3.2. Model execution

The first service provided by mJADES is the execution of multiple simulations in the cloud. This service entails a simple workflow, whose major steps are outlined in the following text:

1. **Model selection**: The first step in the usage of the platform is the selection of a simulation model. The mJADES system maintains a repository of models available for its users. New models can be submitted through a dedicated service (described in the next subsection).
2. **Job submission**: Once the user has selected a model, a customized web interface for that model is generated by the system. mJADES users interact with the system through the web interface to describe a *simulation job*. This is a collection of independent *simulation tasks*, each of which corresponds to a single simulation. Different job specifications will result in different simulation tasks to be executed. The current mJADES prototype offers two simple ways to describe a job as a collection of different tasks: *Parameter Sweep* and *Replications*. The first technique describes a job through a range of values for a given parameter, the latter asks for a given number of repetitions of the simulation (it is useful for statistical validation). It is also possible to specify jobs that use a mixture of both techniques.
3. **Generation of tasks**: After the job is submitted, the mJADES Manager decomposes the job into a collection of tasks. Each task corresponds to a distinct tuple of parameter values and

entails running a (single) simulation of the chosen JADES model parameterized with these values. All the tasks are then made available for execution.

4. **Execution of tasks**: The Core cloudlet is able to fetch a simulation task to perform the simulation and to store the results in a dedicated repository. Multiple instances of the Core cloudlet can be concurrently in execution, executing multiple tasks at once.

5. **Analysis of results**: After the Core cloudlets have completed their execution, the Analyzer fetches the results and generates a web page with statistical or graphical reports (according to the option chosen by the user at job submission time).

### 3.3. Model submission

The model execution service makes it easy for parallel program developers to describe simulation jobs that make very complex evaluations, including statistical validation (through replication) and comparison of design choices (through parameter sweep). Another key function of the system is the ability to support an open repository of models, which can be submitted by the users and stored in the system for successive use. To provide a model repository, it is necessary to offer to developers a practical way to specify the model metadata, such as the model description, its input parameters, and its interface with the user. We have enriched the JADES library with a set of *annotations* that can be used to insert the required metadata directly into the simulation models. The supported annotations are as follows:

- **@SimModel** is used to annotate the model class and allows the modeler to associate to the model useful information as a name, a domain, and a meaningful natural-language description.
- **@SimParam** marks a field in a Java class as a simulation parameter. This annotation is used to identify the inputs of the model and their type.
- **@SimResult** can be used to mark a field that contains a simulation result. This annotation is used to identify the outputs of the model and their type.

mJADES also contains an annotation processor that is able to parse the model code, to recover the annotations, and to use them to understand the structure of the model. More detailed, the SimModel annotation is used to show to users a detailed description of the model in the web interface; the SimParam annotation is used to automatically generate the web form that gets the simulation input; the SimResult interface is used to present a list of selectable results that the user can visualize or send to the Analyzer for further processing.

## 4. MODELING PARALLEL PROGRAMS IN MJADES

In the previous section, we have discussed the operation of the mJADES platform. In this section, we will discuss how parallel programmers can develop models to leverage the platform for their performance prediction and evaluation tasks.

Our modeling framework adopts the layered approach shown in Figure 1. At the lowest layer, the JADES simulation library provides support for process-oriented simulation models described in Java language. JADES has been thoroughly presented in [16]; in brief, it allows the modeler to describe simulation models made of interacting processes that evolve in time by discrete events, which can be the voluntary suspension for a given time (*hold* operation), the use of resources, or the exchange of messages. A resource can be a *facility*, which represents all-or-none resources like servers, a *storage*, which models partially-allocatable resources like memories and disks, or a *buffer*, through which processes can communicate and synchronize by producing and consuming items. A *mailbox* makes it possible for processes to exchange messages. Finally, an *event* can be used for conditional process synchronization: one or more processes wait for events to occur, and other processes declare events as occurred.

On the top of these core components, we have developed a library of higher-level components that simplify the development of performance models for distributed memory, message-passing parallel
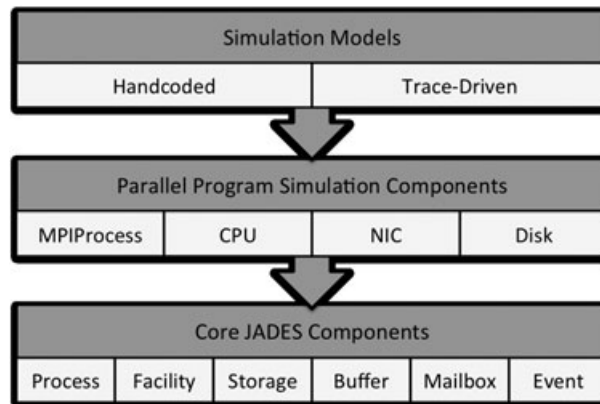
Figure 1. The modeling framework.

programs. These components make use of the core components to represent entities with behaviors commonly found in parallel programs. They can be thought of as building blocks that can be composed to realize full-blown parallel program models. The components that are representative of computational resources are:

- *CPU*, which is a special facility with customizable scheduling and performance-related parameters (clock frequency, processing rate in flops);
- *Network interface controller (NIC)*, which encompasses both a processing facility (that accounts for message processing on the NIC) and a mailbox for communication;
- *Disk*, a kind of storage with customizable performance parameters (data transfer rate and access time).

Other components that have been developed are extensions of the process abstraction. They provide additional functionality to the basic simulation processes. For example, the *MPI Process* shown in the figure - and used in the case study of the next section - adds the following four entities to the basic JADES process:

- *computation*, that is, the use of a CPU facility. A computation results in a possible queuing for the use of the CPU resource, and a hold operation modeling the time spent computing. There are several different options to define the computation time for a given section of code. In our models, we use absolute time measurements or parametric models that predict actual computation time as a function of the sequence of instructions executed. The CPU facility allows for the modeling of contention between multiple processes running on the same processor.
- *communication*, that is, the sending of a message through a network connecting two NICs. In most of our models, communication time is computed by means of a simple linear model based on latency and bandwidth, even if the framework makes it possible to adopt more sophisticated models (e.g., full TCP/IP simulation, or LogP models);
- *input/output (I/O)*, that is, the use of an I/O device. Whenever the effect of I/O is negligible as compared to computation and communication, as in most scientific codes, I/O is not usually modeled.
- program execution *statistics collection*. Low-level simulation statistics (e.g., facility and mailbox usage) are used to compute higher-level figures, more strictly related to parallel program execution, such as CPU usage (for each available CPU), fraction of time spent in communication, total number of bytes exchanged, and total execution time of the simulated application.

The building blocks provided can be composed into a model by means of different techniques. Currently, we support handcoded and trace-driven modeling. Using *handcoding*, the model can be

set up by writing a simulation code on the top of the JADES application programming interface (API) and exploiting the special components described earlier in the text. This technique is the most flexible, but is time-consuming, as it requires a direct programming effort from the model developer. Section 5 describes the construction of a handcoded Jacobi model. The writing of a handcoded model can be performed at a higher level of abstraction by using a suitable metalanguage that hides the details of the simulation application programming interface. One such language is MetaPL, [48] an XML-based language proposed by our research group for the description of parallel programs.

In *trace-driven* simulation, a model is not able *per se* to reproduce the modeled program behavior. It is instead a sort of *player*, driven by the punctual program behavior in time, recorded during actual program execution in the form of *traces*. Tracing can be performed at different levels of granularity. In the case of message-passing parallel code (e.g., for MPI applications), it is typically sufficient to record the relevant computation and communication events. It is important to point out that the model is just a player fed with data from a previous run, but it is parameterized. In theory, a trace should allow to reproduce just one particular program run, the one where the trace was initially recorded. In practice, for most program codes, the availability of a trace makes it possible to predict with sufficient accuracy the program behavior under different working conditions to answer to 'what if' questions (e.g., using faster processors, more memory, a different communication network, ...). A different (and much more complicated) matter is deriving from a recorded trace another trace valid for a different (typically, a higher) number of processes/processors. Trace extrapolation [49, 50] is the derivation of synthetic and parametric traces from actually recorded ones. This makes it possible to study the program behavior under significantly different conditions (in practice, for a higher number of processes/processors) and helps to reduce the size of the recorded execution traces (huge traces are a problem in the case of long-running programs). We provide a predefined model driven by traces, together with a library based on the PMPI profiling interface that can record traces during the actual execution of MPI programs.

Automated model development methods are a hot research topic. In addition to the aforementioned trace extrapolation, another approach recently considered is the use of compile-time analyses. Compiler-derived techniques leverage on static code analysis to characterize the computational program phases. [51, 52] Another possibility is deriving the model (either in source language or in XML interchange language) by applying source-to-source transformations. We are currently working on compile-time analysis and trace extrapolation, which will be supported in prototype form in the future versions of our modeling framework.

## 5. CASE STUDY: PERFORMANCE MODELING OF THE JACOBI ITERATION

This section will show the practical application of mJADES for performance prediction. First, the steps needed to manually derive a performance model for simulation are described using a case study on parallel Jacobi iteration code. Then, the simulation execution and the results obtained through the performance model are shown, discussing the practical validity of the concurrent execution of the simulation workload on computing resources hosted in a cloud.

### 5.1. Parallel algorithm description

The Jacobi method is an iterative solver used to compute a solution of the 2D Laplace equation by the finite differences method. [53] Given a 2D grid representing the region of interest, a stencil code can be used to update the interior points of the grid to the average of neighboring points until convergence is reached. A parallel implementation of the algorithm can be easily constructed by assigning rows of the grid to cooperating processes. To perform the stencil computation, each process needs two boundary rows not belonging to its subdomain (the last row of the preceding process and the first row of the following process). So each process allocates space for its rows and a couple of 'ghost' rows, which mirror the content of the boundary rows residing on neighbor processes. The updated values of these rows must be received at every iteration. It is worth noting that even if the parallel Jacobi iteration is not the best algorithm for the numerical solution of systems of linear

equations, its structure (row-wise decomposition plus ghost rows exchange) is very similar to the one used in more efficient methods, as conjugate gradient or multigrid.

The structure of the algorithm is made of the following operations, repeated in a loop:

- Communicate ghost rows: The processes exchange boundary rows to be able to compute their internal rows. This is done through *send* and *receive* message-passing primitives.
- Local computation: Processes update the internal points; a helper grid is used to store the new values. At the end of the iteration, the helper grid is used to update the main grid and the local error is computed.
- Global error computation: All local errors are compared to find the maximum error. If the value of the global error is under a given threshold, the processes terminate, otherwise a new iteration is started. To compute the global error, an *allreduce* collective operation using the maximum function is used.

### 5.2. Deriving the simulation model

Starting from the informal specification presented earlier in the text, the Jacobi simulation model can be derived by modeling the three steps of the algorithm. To model the computation cost, a regression model has been constructed from measurements of the execution time of sample iterations of the real MPI implementation for variable number of processes. Whenever an implementation of the algorithm is not already available (e.g., at the early stages of program design), synthetic code can be used to obtain a reasonable estimate of computation costs. The predicted time can thus be used as parameter of the *computation* behavior defined in the library components. To model communication, we can replace the communication calls in the algorithm with the corresponding simulated versions provided by the library. Listing 1 shows the code executed by every simulated process in the model.

The MPI calls that can be seen in the model are the aforementioned simulated calls: they do not perform computations as their counterparts in real programs but just account for the elapsing of an amount of simulated time (hopefully) equal to the actual one. Similarly, the *compute* function is used to make simulated time advance as in a real computation on the CPU.

The aforementioned code represents the 'blueprint' of the simulation processes used in the Jacobi Model. As in JADES, a model is a collection of simulation processes, the developer must also specify the model initialization code shown in Listing 2. This code defines the model specification: it must indicate a SimModel annotation, used to generate a description of the model for its users, and a run method that performs the instantiation of the simulation processes. In addition, as shown in Listing 3, the developer should describe the input parameters of the simulation by marking the corresponding fields in his classes with the SimParam annotation, used to automatically generate the code for the web interface. The model is then ready to be submitted to the system, augmenting the model repository maintained by the mJADES Manager.

### 5.3. Running the simulation model

To execute this simulation model, the user can connect to the web interface exposed by the Manager. The first interaction with the user is the model selection form. When the user selects the model for execution, the system updates the interface as shown in Figure 2, adding the model-specific fields to receive user input. These fields are the ones marked with the SimParam annotation described in section 3.3.

In this case study, the performance analysis carried out on the mJADES platform has focused on two main issues:

- assessing the effect of the number of processes into which the application is decomposed (each process is supposed to be allocated on a dedicated CPU);
- assessing the effect of the interconnect, switching between a Fast Ethernet (100 Mbit/s) and a Myrinet. [54]

Listing 1: Jacobi Simulation Model - Process code

```java
import it.unisannio.ing.perflab.jades.core.Process;
import it.unisannio.ing.perflab.jades.resources.Facility;
import it.unisannio.ing.perflab.components.Host;
import it.unisannio.ing.perflab.pps.components.MPIProcess;


public class JacobiIterationProcess extends MPIProcess implements
    Serializable{

  public static int MASTER = 0;
  private int rank;
  private int ROWS, COLS;
  private int ITERS;

  public void run(){
    int numTasks  = MPI_COMM_SIZE();
    int rank = MPI_COMM_RANK(MPI_COMM_WORLD);
    int left = (rank > 0)? rank-1: MPI_PROCESS_NULL;
    int right = (rank < numTasks -1)? rank+1: MPI_PROCESS_NULL;
    for(int i = 0; i < ITERS; i++){
      /*Boundary rows exchange */
      MPI_Request s1 = MPI_ISEND(MPI_COMM_WORLD, left, COLS*8, s1);
      MPI_Request r1 = MPI_IRECEIVE(left, COLS*8, r1);
      MPI_Request s2 = MPI_ISEND(MPI_COMM_WORLD, right, COLS*8, s2);
      MPI_Request r2 = MPI_IRECEIVE(right, COLS*8, r2);
      MPI_WAIT(s1);
      MPI_WAIT(r1);
      MPI_WAIT(s2);
      MPI_WAIT(r2);
      /*Local computation*/
      this.compute(0.1122* Math.pow(numTasks,-1.002));
      /*Time to copy from helper to main matrix*/
      this.compute(0.0341* Math.pow(numTasks,-0.991));
      //Global error reduction
      MPI_ALLREDUCE(MPI_COMM_WORLD, 1 * 8);
    }
  }
}
```

Listing 2: Jacobi Simulation Model - Initialization

```java
@SimModel(modelName = "Jacobi Model", packageName = "", description = "
    Models an MPI Jacobi Iteration")
public class JacobiModel extends Model implements Serializable{

  private SimulationConfig config;

  public JacobiModel(SimulationConfig config) {
    this.config = config;
  }

  public void run() {
    Host[] hosts = config.getHosts();
    int numProcesses = config.getNumProcesses();
    String[] machineFile = config.getMachineFile();
    start(new JacobiLauncherProcess("JacobiLauncher", numProcesses, hosts,
        machineFile));
  }
}
```

Listing 3: Jacobi Simulation Model - Input Parameters

```java
package config;

import interfaces.SimParam;
import components.Host;

public class SimulationConfig {
    @SimParam(paramName="numProcesses")
    private int numProcesses;

    @SimParam(paramName="hosts")
    private String hosts;

    @SimParam(paramName="machineFile")
    private String machineFile;

    //...getters and setters follow...
}
```
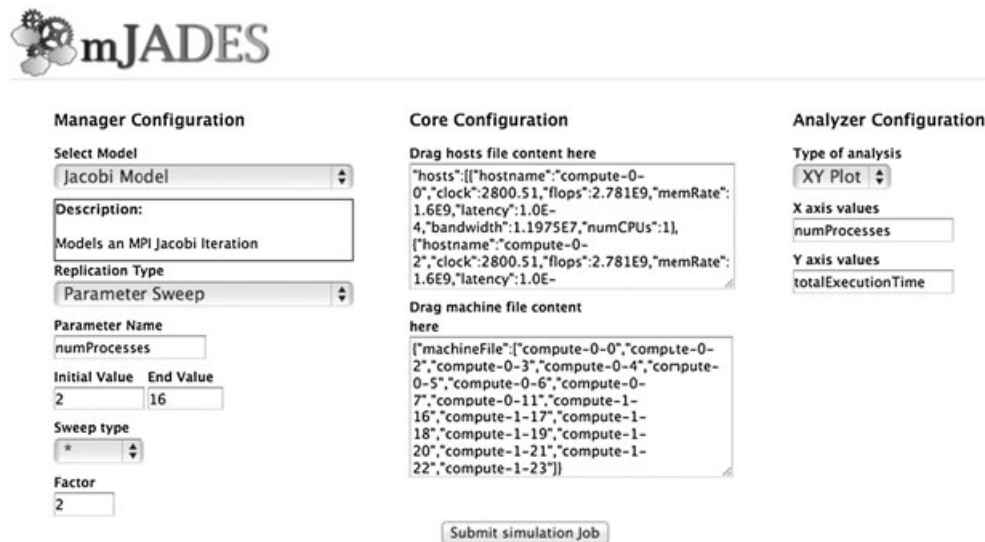


Figure 2. mJADES web interface.

In all the experiments, the problem grid size was fixed in the model to $2000 \times 2000$ points. For the first analysis, we chose a sweep over the $nP$ parameter, which represents the number of simulated processes. When the user submits this request through the web interface, a JavaScript Object Notation (JSON) message describing the job is generated, as shown in Listing 4. It is worth pointing out that the message is produced automatically by scripts in the web interface (expressed through the jQuery language) and it is completely hidden from the developer. Then this message is received by the Manager, which starts the execution workflow described in section 3.2.

For the second analysis, a sweep was performed over a compound parameter representing latency and bandwidth of the interconnection network. The values of these parameters for Fast Ethernet and Myrinet networks were obtained on our PoweRcost cluster through benchmarking. The two interconnects were simulated for every number of processes in the range of interest. For brevity, we do not show here the simulation input and the web interface, as they are similar to the previous ones.

### 5.4. Experimental results

An experimental mJADES platform has been deployed on a private Eucalyptus cloud composed of 16 physical nodes hosting the virtual machines on which cloudlets are deployed. One node hosts the
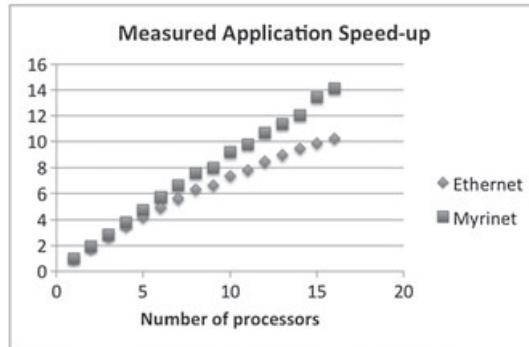
**Listing 4: JSON Configuration Message**

```
{
    "modelName":"MPISimulator",
    "nOfReplications":1,
    "sweeps":[
        {"paramName":"numProcesses",
        "sweepType":"*",
        "initialValue":2,
        "endValue":16
    }]
    hosts: ...
}
```
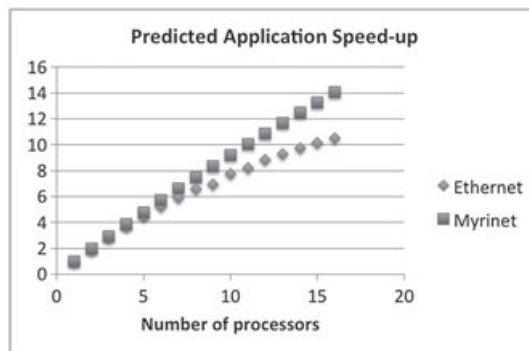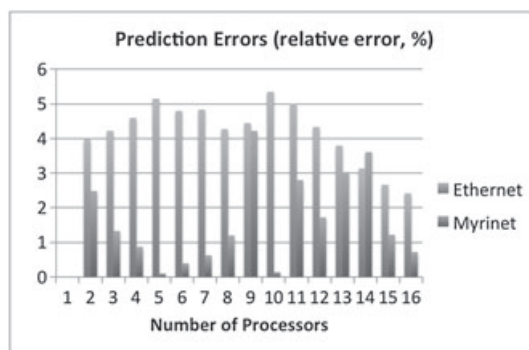


(a) Predicted speed-up



(b) Measured speed-up



(c) Prediction Error

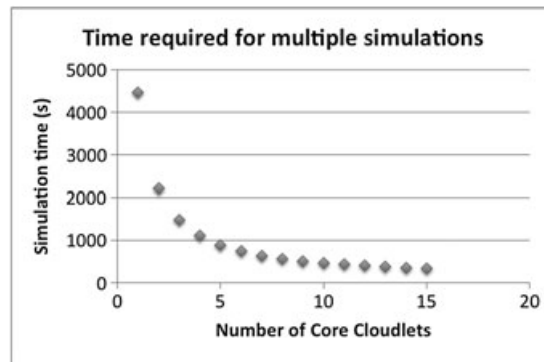Figure 3. Results for the Jacobi model case study.

Figure 4. Simulation time and speedup versus number of cloudlets used.

web server and the mJADES manager cloudlet, whereas the others are dedicated to host instances of the mJADES core cloudlets. For simplicity's sake, in this study, we have not investigated the effect of oversubscribing the processors, running just one cloudlet for each CPU core.

Even if the objective of this paper is to stress the ease of use and the fast response times of concurrent simulation in a cloud, it is nevertheless interesting to present the simulation results and to compare them to the actual results measured on the real target machine. We can not refrain from pointing out once again that simulation results are practically indistinguishable from actual measurements, and so that performing long and exhausting performance testing on the target machine is often just a waste of time. Figure 3 shows the plots of the speedup of an MPI version of the code of the Jacobi iteration, as predicted by simulation (Figure 3(a)) and as measured on the test machine (Figure3(b)). Prediction errors are in Figure 3(c): the maximum relative error throughout the range from 2 to 16 processes is 5.35%.

Simulation can be very effective but, on the bad side, launching a large number of simulation sessions for different values of the parameters can be as frustrating as running tests on the real hardware (the latter requires hardware availability, in addition). This is the point where the almost unlimited resources of clouds and mJADES come into play. To evaluate the benefit of running multiple simulations concurrently, multiple tests have been run with the mJADES manager configured to use a varying number of instances of the Core cloudlet. Figure 4 shows the effect of running one of the composite simulation jobs, the sweep over the number of processes, on a number of Core cloudlets ranging from 1 to 15. The sweep varies $nP$ from 10 to 510, at increments of 25. This produces a total of 20 simulations with progressively growing execution time. To help load balancing, the tasks are handed from the largest (the one for the highest value of $nP$) to the smallest. Running all the 20 simulations on the same Core cloudlet requires about 4500 seconds of simulation time, while as the number of cloudlets grows, the time drops to 330 seconds, leading to a near linear speedup even if a minor imbalance is still present. Furthermore, the overhead introduced by the workload distribution mechanism of mJADES is almost negligible. In our opinion, this is an interesting result, showing the substantial validity of the concurrent simulation approach, which spreads beyond the field of the application presented here as an example.

## 6. CONCLUSIONS AND FUTURE WORK

The objective of this paper has been to show that finally the use of simulation techniques and tools for parallel program performance prediction has become viable. We have discussed the problems linked to their use to obtain quantitative performance evaluations for multiple working conditions at once. We claim that the use of mJADES, which allows concurrent simulation runs on resources automatically leased from a cloud, solves the majority of the issues that in the last decade has prevented the widespread diffusion of performance prediction for parallel program development and analysis.

The program modeling process currently is made less complex by the availability of the simulation components developed for mJADES; on the other hand, the model annotations simplify the deployment. We have presented an experimentation on a simple but not trivial message-passing code. Our tests show that the concurrent simulation approach that is the distinctive feature of mJADES can be very useful, making it possible to automatize and to speed up considerably the launch of a bag of simulation tasks that would otherwise require long waiting times. The accuracy of the performance prediction that can be obtained by simulation (errors smaller than 5.35%) is not an absolute novelty but deserves a mention. Perhaps more important here is the possibility opened by mJADES to execute a huge number of simulation tasks in parallel with little or no user intervention. Our future work will try to automatize the software modeling step and to further optimize the mJADES prototype used for the experiments presented in this paper.

## REFERENCES

1. Smith CU. Introduction to software performance engineering: Origins and outstanding problems. In *SFM*, Vol. 4486, Bernardo M, Hillston J (eds), Lecture Notes in Computer Science. Springer: Berlin Heidelberg, 2007; 395–428.
2. Smith CU, Williams LG. Introduction to software performance engineering. *Int. CMG Conference,* Computer Measurement Group, Las Vegas, Nevada, USA, 2004; 853–864.
3. Balsamo S, Marzolla M. A simulation-based approach to software performance modeling. In *Proceedings of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-11. ACM: New York, NY, USA, 2003; 363–366, DOI: 10.1145/940071.940122.
4. López-Grao JP, Merseguer J, Campos J. From UML activity diagrams to Stochastic Petri nets: Application to software performance engineering. *SIGSOFT Software Engineering Notes* 2004; **29**(1):25–36. DOI: 10.1145/974043.974048.
5. Becker S, Koziolek H, Reussner R. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software* 2009; **82**(1):3–22. DOI: 10.1016/j.jss.2008.03.066.
6. Aversa R, Mazzeo A, Mazzocca N, Villano U. Heterogeneous system performance prediction and analysis using PS. *IEEE Concurrency* 1998; **6**(3):20–29.
7. Mazzocca N, Rak M, Villano U. MetaPL: A notation system for parallel program description and performance analysis. In *Parallel Computing Technologies*, Vol. 2127, Malyshkin VE (ed.), Lecture Notes in Computer Science. Springer-Verlag: Berlin Heidelberg, 2001; 80–93.
8. Aversa R, Di Martino B, Rak M, Venticinque S, Villano U. Performance prediction through simulation of a hybrid MPI/openMP application. *Parallel Computing* 2005; **31**(10–12):1013–1033.
9. Mancini E, Villano U, Mazzocca N, Rak M, Torella R. Performance-driven development of a web services application using MetaPL/HeSSE. In *Proceedings of the 13th Euromicro Conference on Parallel and Distributed Processing*. IEEE Computer Society: Washington, DC, USA, 2005; 12–19.
10. Di Martino B, Mancini E, Rak M, Torella R, Villano U. Cluster systems and simulation: From benchmarking to off-line performance prediction. *Concurrency and Computation: Practice and Experience* 2007; **19**(11):1549–1562.
11. Aversa R, Di Martino B, Rak M, Venticinque S. In *Performance Prediction for HPC on Clouds*, chap. 17. John Wiley & Sons, Inc., 2011; 437–456, DOI: 10.1002/9780470940105.ch17.
12. Cuomo A, Rak M, Villano U. mJADES: Concurrent simulation in the cloud. *2nd International Workshop on Intelligent Computing at Large Scale*, Palermo, Italy, July 4–6, 2012; 853–860, DOI: 10.1109/CISIS.2012.134.
13. Fujimoto RM. Parallel discrete event simulation. *Communications of the ACM* 1990; **33**:30–53.
14. Perumalla KS. Parallel and distributed simulation: Traditional techniques and recent advances. *Proceedings of the 38th Conference on Winter Simulation,* Winter Simulation Conference, New Orleans, LA, 2006; 84–95.
15. Fujimoto RM. Distributed simulation systems. *Proceedings of the 35th Conference on Winter Simulation: Driving Innovation,* WSC '03, Winter Simulation Conference, New Orleans, LA, 2003; 124–134.
16. Cuomo A, Rak M, Villano U. Process-oriented discrete-event simulation in Java with continuations: Quantitative performance evaluation. In *Proceedings of the International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*. SciTePress: Setubal, Portugal, 2012; 87–96.
17. Messina F, Pappalardo G, Santoro C. ComplexSim: An SMP-aware complex network simulation framework. In *2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems*. IEEE: Washington, DC, USA, 2012; 861–866.

18. Pelkey J, Riley G. Distributed simulation with MPI in ns-3. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering): Brussels, Belgium, 2011; 410–414.

19. Chandy KM, Misra J. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering* 1979; **SE-5**(5):440–452.

20. Jefferson D, Sowizral H. Fast concurrent simulation using the time warp mechanism. *SCS Conf. Distributed Simulation*, San Diego, CA, 1985; 63–69.

21. Castilla I, Aguilar RM, Callero Y. Java for parallel discrete event simulation: A survey. *International Journal of Simulation and Process Modelling* Jan 2011; **6**(4):250–260. DOI: 10.1504/IJSPM.2011.048006.

22. Heidelberger P. Statistical analysis of parallel simulations. In *Proceedings of the 18th Conference on Winter Simulation*, WSC '86. ACM: New York, NY, USA, 1986; 290–295.

23. Marr C, Storey C, Biles WE, Kleijnen JPC. A Java-based simulation manager for web-based simulation. *Simulation Conference, 2000. Proceedings. Winter*, Vol. 2, Orlando, FL, 2000; 1815 –1822, DOI: 10.1109/WSC.2000.899174.

24. Healy KJ, Kilgore RA. Introduction to SILK and Java-based simulation. In *Proceedings of the 30th Conference on Winter Simulation*, WSC '98. IEEE Computer Society Press: Los Alamitos, CA, USA, 1998; 327–334.

25. Seo C, Han Y, Lee H, Jung J, Lee C. Implementation of cloud computing environment for discrete event system simulation using service oriented architecture. In *2010 IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing (EUC)*. IEEE: Washington, DC, USA, 2010; 359–362.

26. Guo S, Bai F, Hu X. Simulation software as a service and service-oriented simulation experiment. In *IEEE International Conference on Information Reuse and Integration (IRI)*. IEEE: Las Vegas, Nevada, USA, 2011; 113–116.

27. Tsai W, Li W, Sarjoughian H, Shao Q. SimSaaS: Simulation software-as-a-service. In *Proceedings of the 44th Annual Simulation Symposium*. Society for Computer Simulation International: San Diego, CA, USA, 2011; 77–86.

28. Park A, Fujimoto R. A scalable framework for parallel discrete event simulations on desktop grids. In *2007 8th IEEE/ACM International Conference on Grid Computing*. IEEE: Washington, DC, USA, 2007; 185–192.

29. Fujimoto R, Malik A, Park A. Parallel and distributed simulation in the cloud. *SCS M&S Magazine* 2010; **3**.

30. D'Angelo G. Parallel and distributed simulation from many cores to the public cloud. In *2011 International Conference on High Performance Computing and Simulation (HPCS)*. IEEE: Washington, DC, USA, 2011; 14–23.

31. Taylor S, Ghorbani M, Kiss T, Farkas D, Mustafee N, Kite S, Turner S, Strassburger S. Distributed computing and modeling & simulation: Speeding up simulations and creating large models. *Proceedings of the 2011 Winter Simulation Conference*, Phoenix, AZ, 2011; 161–175.

32. von Laszewski G, Diaz J, Wang F, Fox G. Comparison of multiple cloud frameworks. In *2012 IEEE 5th International Conference on Cloud computing (CLOUD)*, IEEE: Washington, DC, USA, 2012; 734–741.

33. Mancini E, Rak M, Villano U. Perfcloud: GRID services for performance-oriented development of cloud computing applications. In *18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, WETICE 2009, Groningen, the Netherlands, 29 june - 1 july 2009, Proceedings*, Reddy S (ed.). IEEE Computer Society: Washington, DC, USA, 2009; 201–206.

34. Tusa F, Paone M, Villari M, Puliafito A. CLEVER: A cloud-enabled virtual environment. *2010 IEEE Symposium on Computers and Communications (ISCC)*, Riccione, Italy, 2010june; 477 –482, DOI: 10.1109/ISCC.2010.5546555.

35. Petcu D, Macariu G, Panica S, Craciun C. Portable cloud applications-from theory to practice. *Future Generation Computer Systems* 2012. DOI: 10.1016/j.future.2012.01.009. (Available from: http://www.sciencedirect.com/science/article/pii/S0167739X12000210) [Accessed on 7 January 2013], In press.

36. Petcu D, Crăciun C, Neagul M, Panica S, Di Martino B, Venticinque S, Rak M, Aversa R. Architecturing a sky computing platform. In *Towards a Service-Based Internet. Servicewave 2010 Workshops*. Springer: Berlin Heidelberg, 2011; 1–13.

37. Petcu D, Craciun C, Neagul M, Lazcanotegui I, Rak M. Building an interoperability API for sky computing. In *2011 International Conference on High Performance Computing and Simulation (HPCS)*. IEEE: Washington, DC, USA, 2011; 405–411.

38. Petcu D, Craciun C, Rak M. Towards a cross platform cloud API. Components for cloud federation. In *Procs. 1st International Conference on Cloud Computing and Services Science*. Scitepress–Science and Technology publications: Setubal, portugal, 2011; 166–169.

39. Prakash S, Bagrodia R. MPI-SIM: Using parallel simulation to evaluate MPI programs. In *Proceedings of the 30th Conference on Winter Simulation*. IEEE Computer Society Press: Los Alamitos, CA, USA, 1998; 467–474.

40. Badia R, Labarta J, Gimenez J, Escale F. Dimemas: Predicting MPI applications behavior in grid environments. *Workshop on Grid Applications and Programming Tools (GGF8)*, Vol. 86, Seattle, WA, USA, 2003; 52–62.

41. Zhai J, Chen W, Zheng W. Phantom: Predicting performance of parallel applications on large-scale parallel machines using a single node. *ACM Sigplan Notices* 2010; **45**(5):305–314.

42. Achour S, Ammar M, Khmili B, Nasri W. MPI-PERF-SIM: Towards an automatic performance prediction tool of MPI programs on hierarchical clusters. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*. IEEE: Washington, DC, USA, 2011; 207–211.

43. Rodrigues A, Hemmert K, Barrett B, Kersey C, Oldfield R, Weston M, Risen R, Cook J, Rosenfeld P, Cooper-Balis E, Jacob B. The structural simulation toolkit. *ACM SIGMETRICS Performance Evaluation Review* 2011; **38**(4): 37–42.

44. Clauss P, Stillwell M, Genaud S, Suter F, Casanova H, Quinson M. Single node on-line simulation of MPI applications with SMPI. In *2011 IEEE International Parallel & Distributed Processing Symposium (IPDPS)*. IEEE: Washington, DC, USA, 2011; 664–675.

45. Susukita R, Ando H, Aoyagi M, Honda H, Inadomi Y, Inoue K, Ishizuki S, Kimura Y, Komatsu H, Kurokawa M, Murakami K, Shibamura H, Yamamura S, Yu Y. Performance prediction of large-scale parallell system and application using macro-level simulation. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. IEEE Press: Piscataway, NJ, USA, 2008; 20.

46. Zheng G, Kakulapati G, Kalé L. Bigsim: A parallel simulator for performance prediction of extremely large parallel machines. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE: Washington, DC, USA, 2004; 78.

47. Cuomo A, Rak M, Villano U. Cloud-based concurrent simulation at work: Fast performance prediction of parallel programs. *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 21th IEEE International Workshops on - Toulouse, 25-27 june 2012*, 2012; 137–142, DOI: 10.1109/WETICE.2012.74.

48. Mazzocca N, Rak M, Villano U. MetaPL: A notation system for parallel program description and performance analysis. In *Parallel Computing Technologies*, Vol. 2127, Malyshkin V (ed.), Lecture Notes in Computer Science. Springer-Verlag: Berlin Heidelberg, 2001; 80–93.

49. Hoefler T, Schneider T, Lumsdaine A. Loggopsim-simulating large-scale applications in the loggops model. *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC*, Vol. 10, Chicago, IL, USA, 2010; 597–604.

50. Wu X, Mueller F. Scalaextrap: Trace-based communication extrapolation for SPMD programs. *SIGPLAN Notices* 2011; **46**(8):113.

51. Cascaval C, DeRose L, Padua D, Reed D. Compile-time based performance prediction. In *Languages and Compilers for Parallel Computing*, Vol. 1863. Springer: Berlin Heidelberg, 2000; 365–379.

52. Marin G, Mellor-Crummey J. Cross-architecture performance predictions for scientific applications using parameterized models. *ACM SIGMETRICS Performance Evaluation Review,* ACM 2004; **32**(1):2–13.

53. Andrews G. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison-Wesley: Boston, MA, USA, 2002.

54. Boden N, Cohen D, Felderman R, Kulawik A, Seitz C, Seizovic J, Su W. Myrinet: A gigabit-per-second local area network. *IEEE Micro* 1995; **15**(1):29–36.