# Toward Better Simulation of MPI Applications on Ethernet/TCP Networks

Paul Bédaride*, Augustin Degomme†, Stéphane Genaud‡, Arnaud Legrand†,
George S. Markomanolis§, Martin Quinson*, Mark Stillwell¶, Frédéric Suter‖, Brice Videau†

* Loria/INRIA/Univ. of Nancy. Nancy, France. Email: firstname.lastname@loria.fr
† CNRS/INRIA/Univ. of Grenoble. Grenoble, France. Email: firstname.lastname@imag.fr
‡ University of Strasbourg. Strasbourg, France. Email: genaud@unistra.fr
§ INRIA, LIP, ENS Lyon. Lyon, France. Email: gmarko01@ens-lyon.fr
¶ Engineering Computing. Cranfield University, U.K. Email: m.stillwell@cranfield.ac.uk
‖ IN2P3 Computing Center, CNRS. Lyon-Villeurbanne, France. Email: fsuter@cc.in2p3.fr

***Abstract*—Simulation and modeling for performance prediction and profiling is essential for developing and maintaining HPC code that is expected to scale for next-generation exascale systems, and correctly modeling network behavior is essential for creating realistic simulations. In this article we describe an implementation of a flow-based hybrid network model that accounts for factors such as network topology and contention, which are commonly ignored by other approaches. We focus on large-scale, Ethernet-connected systems, as these currently compose 37.8% of the top500 index, and this share is expected to increase as higher-speed 10 and 100GbE become more available. The European Mont-Blanc project, a prototype exascale platform concerned with power-efficiency, will also use Ethernet-based interconnect. Our model is implemented within SMPI, an open-source MPI implementation that connects real applications to the SimGrid simulation framework. SMPI provides implementations of collective communications based on current versions of both OpenMPI and MPICH. SMPI and SimGrid also provide methods for easing the simulation of large-scale systems, including shadow execution, memory folding, and support for both online and offline (i.e., post-mortem) simulation. We validate our proposed model by comparing traces produced by SMPI with those from real world experiments, as well as with those obtained using other established network models. Our study shows that SMPI has a consistently better predictive power than classical LogP-based models for a wide range of scenarios including both established HPC benchmarks and real applications.**

## I. Introduction

In the High Performance Computing (HPC) field, accurately predicting the execution time of parallel applications is of utmost importance to assess their scalability, and this is particularly true for applications slated for deployment on next-generation exascale systems. While much effort has been put towards understanding the high-level behavior of these applications based on abstract communication primitives, real-world implementations often provide a number of confounding factors that may break basic assumptions and undermine the applicability of these higher level models. For example, implementations of the MPI standard can select different protocols and transmission mechanisms (e.g., eager vs. rendez-vous) depending on message size and network capabilities. Simple delay-based models also do not account for network realities such as network topology and contention. In this work we demonstrate how even relatively minor deviations in low-level implementation can adversely affect the ability of simulations to predict real-world performance, and propose a new network model that extends previous approaches to better account for topology and contention in high-speed TCP networks. We focus on large-scale, Ethernet-connected systems, which currently compose 37.8% of the top500 index [41]. This share is only expected to increase as higher-speed 10 and 100GbE become more available. The European Mont-Blanc project, a prototype exascale platform that focuses on power-efficiency, will also use Ethernet-based interconnect [30].

Packet-level simulation has long been considered the "gold standard" for modeling network communication, and is available for use in a number of simulation frameworks [33], [47]. However, there are a number of reasons to consider alternatives when simulating parallel applications. First, such applications are likely to generate large amounts of network traffic, and packet-level simulation has high overheads, resulting in simulations that may take significantly longer to run than the corresponding physical experiments. Second, implementing packet-level simulations that accurately model real world behavior requires correctly accounting for a vast array of factors, such as the precise routing algorithms implemented within operating system kernels [28]. In practice, there is little difference between an inaccurate model and an accurate model of the wrong system.

When packet-level simulation becomes too costly or intractable, the most common approach is to resort to simpler delay models that ignore the network complexity. Among these models, the most famous are those of the LogP family [13], [1], [27], [26]. The LogP model was originally proposed by Culler et al. [13] as a realistic model of parallel machines for algorithm design. It was claimed as more realistic than more abstract models such as PRAM or BSP. This model captures key characteristics of real communication platforms while remaining amenable to complexity analysis. Unfortunately, while this model may reflect the behavior of specialized HPC networks from the early 1990s, it ignores potentially confounding factors present in modern-day systems.

Flow-based models are a reasonable alternative to both simple analytic models and expensive and difficult-to-instantiate packet-level simulation. In a flow based model, network traffic

is treated as a steady state fluid flow through interconnected pipes of varying lengths and sizes (representing delay and bandwidth). Flow-based models are computationally tractable while being able to account for factors such as network heterogeneity. We seek to capture the advantages of a flow-based approach by extending existing validated models of point-to-point communication to better account for network topology and message contention. Recent work suggests that well-tuned flow-based simulation may be able to provide reasonably accurate results for less effort than packet-based simulation, and at much lower cost [43].

Our model is implemented within SMPI [12], an open-source MPI implementation that connects real-world applications to the SimGrid [10] simulation framework. With SMPI, standard MPI applications written in C or FORTRAN can be compiled and run in a simulated network environment, and traces documenting computation and communication events can be captured without incurring errors from tracing overheads or improper synchronization of clocks as in physical experiments. SMPI has recently been extended so that the low-level implementations of MPI collective operations more accurately reflect current production versions of both OpenMPI and MPICH. SMPI and SimGrid also provide a number of useful features for simulating applications that may require large amounts of time or system resources to run, including shadow execution, memory folding, and off-line simulation by replay of execution traces [15]. We validate our results by comparing application traces produced by SMPI using our network model with those from real world environments. We also compare with traces obtained using models from the literature.

The specific contributions described in this paper are as follows:

- we propose a new flow-based network model that extends previous LogP-based approaches to better account for network topology and message contention;
- we describe SMPI, the simulation platform, and some useful extensions that we have developed to make this tool more useful to developers and researchers alike (e.g., SMPI now implements **all** the collective algorithms and selection logics of both OpenMPI and MPICH for more faithful comparisons);
- we provide a number of experimental results demonstrating how these extensions improve the ability of SMPI to accurately model the behavior of real-world applications on existing platforms, and also show that competing frameworks using previous models from the literature are unlikely to obtain consistently good results;
- we demonstrate the effectiveness of our proposal by making a thorough study of the validity of our models against hierarchical clusters using TCP over Gigabit Ethernet.

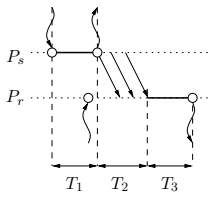
This paper is organized as follows: In Section II we discuss essential background information in the area of network modeling. In Section III we describe SMPI, our chosen implementation framework, along with some key features that make it suitable for simulation of large scale systems, and comparisons to competing simulation platforms. In Section IV we describe our proposed hybrid network model, focusing particularly on how this model captures the complexities of network topology and message contention. In Section V we describe experiments conducted to [in]validate the model, including comparisons of traces from real-world environments as well as traces produced using other network models. In Section VI we demonstrate the capacity of SMPI to simulate complex MPI applications on a single machine. In Section VII we discuss related work, including other frameworks for simulating parallel applications and their relevant drawbacks. We conclude in Section VIII and also provide a description of proposed areas of future work.

## II. Network Modeling Background

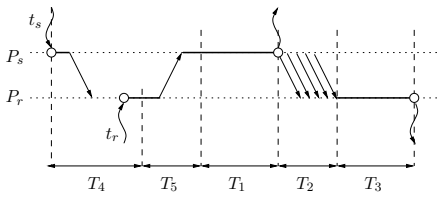
In the LogP model, a parallel machine is abstracted with four parameters: $L$ is an upper bound on the *latency* of the network, i.e., the maximum delay incurred when communicating a word between two machines; $o$ denotes the CPU *overhead*, i.e., the time that a processor spends processing an emission or a reception and during which it cannot perform any other operation; $g$ is the *gap* between messages, whose reciprocal is the processor communication bandwidth; and $P$ represents the number of processors. Assuming that two processors are ready to communicate, the time to transfer a message of size $k$ is then $o+(k-1)\max(g,o)+L+o$. Ideally, these four parameters should be sufficient to design efficient algorithms. Indeed, this model accounts for computation/communication overlap since for *short* messages, the sender is generally released before the message is actually received. Unfortunately, it fails to accurately model the transmission of *long* messages that are common in modern parallel applications.

The LogGP model proposed in [1] introduces an additional parameter $G$ to represent the larger effective bandwidth experienced by long messages. The formula for short messages is unchanged but becomes to $o+(k-1)G+L+o$ for long ones. This simple distinction between short and long messages was extended in [27] with the *parameterized LogP* model in which $L$, $o$, and $g$ depend on the message size. The rationale is that the overall network performance results from complex interactions between the middleware, the operating system, and the transport and network layers. Hence, performance is generally neither strictly linear nor continuous. This model also introduces a distinction between the sender overhead $o_s$ and the receiver overhead $o_r$. However, such models may be difficult to use to design algorithms. For instance, they assume that senders and receivers synchronize and include that synchronization cost in the overhead while some MPI implementations use schemes that may not require synchronization, depending on message size.

Finally, Ino *et al.* proposed in [26] the LogGPS model that extends LogGP by adding two parameters $s$ and $S$ to capture the lack of linearity and the existence of a synchronization threshold. Overheads are represented as affine functions $o+kO_s$ where $O_s$ (resp. $O_r$) is the overhead per byte at the sender (resp. receiver) side. This model is described in Figure 1, where $t_s$ (resp. $t_r$) is the time at which




(a) Asynchronous mode ($k \leq S$).




(b) Rendez-vous mode ($k > S$).

| Routine | Condition | Cost |
|---|---|---|
| `MPI_Send` | $k \leq S$ | $T_1$ |
| | $k > S$ | $T_4 + T_5 + T_1$ |
| `MPI_Recv` | $k \leq S$ | $\max(T_1 + T_2 - (t_r - t_s), 0) + T_3$ |
| | $k > S$ | $\max(o + L - (t_r - t_s), 0) + o +$ $T_5 + T_1 + T_2 + T_3$ |
| `MPI_Isend` | | $o$ |
| `MPI_Irecv` | | $o$ |

(c) MPI routine costs.

$$T_1 = o + kO_s \qquad T_2 = \begin{cases} L + kg & \text{if } k < s \\ L + sg + (k-s)G & \text{otherwise} \end{cases} \qquad T_3 = o + kO_r \qquad T_4 = \max(L + o, t_r - t_s) + o \qquad T_5 = 2o + L$$

Figure 1. The LogGPS model [26] in a nutshell.

`MPI_Send` (resp. `MPI_Recv`) is issued. When the message size $k$ is smaller than $S$, messages are sent asynchronously (Figure 1(a)). Otherwise, a *rendez-vous* protocol is used and the sender is blocked at least until the receiver is ready to receive the message (Figure 1(b)). The $s$ threshold is used to switch from $g$ to $G$, i.e., from short to long messages, in the equation. The message transmission time is thus continuously piece-wise linear in message size (Figure 1(c)).

To summarize, the main characteristics of the LogGPS model are: the expression of overhead and transmission times as **continuous piece-wise linear** functions of message size; accounting for **partial asynchrony** for small messages, i.e., sender and receiver are busy only during the overhead cycle and can overlap communications with computations the rest of the time; a **single-port model**, i.e., a sequential use of the network card which implies that a processor can only be involved in at most one communication at a time; and **no topology** support, i.e., contention on the core of the network is ignored as all processors are assumed to be connected through independent bidirectional communication channels. Most of these hypothesis are debatable for many modern computing infrastructures. For example, with multi-core machines, many MPI processes can be mapped to the same node. Furthermore, the increase in the number of processors no longer allows one to assume uniform network communications. Finally, protocol switching typically induces performance modifications on CPU usage similar to those on effective bandwidth, while only the latter are captured by these models.

One alternative to both expensive and difficult-to-instantiate packet-level models and simplistic delay models is flow-level models. These models account for network heterogeneity and have thus been used in simulations of grid, peer-to-peer, and cloud computing systems. Communications, represented by *flows*, are simulated as single entities rather than as sets of individual packets. The time to transfer a message of size $S$ between processors $i$ and $j$ is then given by $T_{i,j}(S) = L_{i,j} + S/B_{i,j}$, where $L_{i,j}$ (resp. $B_{i,j}$) is the end-to-end network latency (resp. bandwidth) on the route connecting $i$ and $j$. Estimating the bandwidth $B_{i,j}$ is difficult as it depends on the network topology and on interactions with every other flow. This is generally done by assuming that the flow has reached *steady-state*, in which case the simulation amounts to solving a bandwidth sharing problem, i.e., determining how much bandwidth is allocated to each flow. More formally: *Consider a connected network that consists of a set of links $\mathcal{L}$, in which each link $l$ has capacity $B_l$. Consider a set of flows $\mathcal{F}$, where each flow is a communication between two network vertices along a given path. Determine a "realistic" bandwidth allocation $\rho_f$ for flow $f$, such that:*

$$\forall l \in \mathcal{L}, \sum_{f \text{ going through } l} \rho_f \leq B_l \ .$$

Many different sharing methods can be used and have been evaluated (for example, in [43]). While such models are rather flexible and account for many non-trivial phenomena (e.g., RTT-unfairness of TCP or cross-traffic interferences) [43], they ignore protocol oscillations, TCP slow start, and more generally all transient phases between two steady-state operation points as well as very unstable situations. Therefore, they provide generally a very good upper-bound of what can be achieved with a given network, and can serve as a basis on which to build more accurate models.

## III. The SMPI Framework

The goal of our research is to use modeling and simulation to better understand the behavior of real-world large-scale parallel applications, which informs the choice of an appropriate simulation platform. That is, simulations for studying the fine-grain properties of network protocols may have little in common with simulations for studying the scalability of some large-scale parallel computing application. Likewise, models used in algorithm design are expected to be much simpler than those used for performance evaluation purposes. Our choice of SMPI as an implementation environment reflects this goal, as SMPI allows for relatively easy conversion of real-world applications to simulation, and provides a number of useful features for enabling large-scale simulations. SMPI implements about 80% of the MPI 2.0 standard, including most of the network communication related functions, and interfaces directly with the SimGrid simulation toolkit [10]. SimGrid is a *versatile* tool to study the behavior of large-scale distributed systems such as grids, clouds or peer-to-peer systems. It has been shown to be often much more scalable than ad hoc simulators [17], [35] and to handle simulations with up to two millions of processes [35] without resorting to a parallel machine. In this section we describe SMPI in

greater detail, focusing first on its general approach and then later highlighting some of these features.

Full simulation of a distributed application, including CPU and network emulation, induces high overheads, and for many cases it can be even more resource intensive than direct experimentation. This, coupled with the fact that a major goal in many simulations is to study the behavior of large-scale applications on systems that may not be available to researchers, means that there is considerable interest in more efficient approaches. The two most widely applied of these are off-line simulation and partial on-line simulation, both of which are available through SMPI.

In *off-line simulation* or "post-mortem analysis" the application to be studied is instrumented before being in a real-world environment. Data about the program execution, including periods of computation, the start and end of any communication events, and possibly additional information such as the memory footprint at various points in time, is logged to a trace file. These traces can then be "replayed" in a simulated environment, considering different "what-if?" scenarios such as a faster or slower network, or more or less powerful processors on some nodes. This **trace replay** is usually much faster than direct execution, as the computation and communications are not actually executed but abstracted as *trace events*. A number of tools [2], [24], [40], [32], [46], [23], including SMPI, support the off-line approach.

Off-line simulation carries with it a number of caveats: It assumes that programs are essentially deterministic, and each node will execute the same sequence of computation and communication events regardless of the order in which messages are received. A bigger challenge is that it is extremely difficult to predict the result of changing the number of nodes–while there is considerable interest and research in this area [24], [44], [9], the difficulty of predicting the execution profile of programs in general, and the fact that both applications and MPI implementations are likely to vary their behavior based on problem and message size, suggests that reliably guaranteeing results that are accurate within any reasonable bound may be impossible in the general case. Another problem with this approach is that instrumentation of the program can add delays, particularly if the program carries out large numbers of fine-grained network communications, and if this is not carefully accounted for then the captured trace may not be representative of the program in its "natural" state.

By contrast, the on-line approach relies on the execution of the program within a carefully-controlled real-world environment: computational sections are executed in full speed on the available hardware, but timing and delivery of messages are determined by the simulation environment (in the case of SMPI this is provided by SimGrid). This approach is much faster than full emulation (although slower than trace replay), yet preserves the proper ordering of computation and communication events. This is the standard approach for SMPI, and a number of other simulation toolkits and environments also followed this approach [33], [16], [3], [37].

To support simulations at very large scale, SMPI allows for

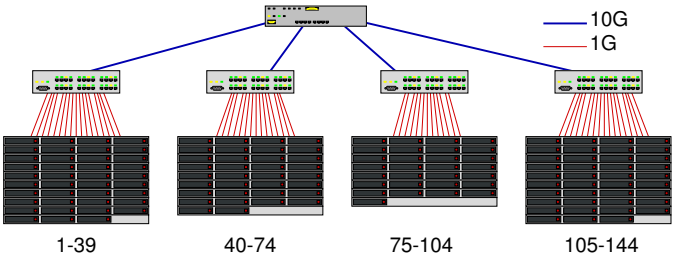


Figure 2. The graphene cluster: a hierarchical Ethernet-based cluster.

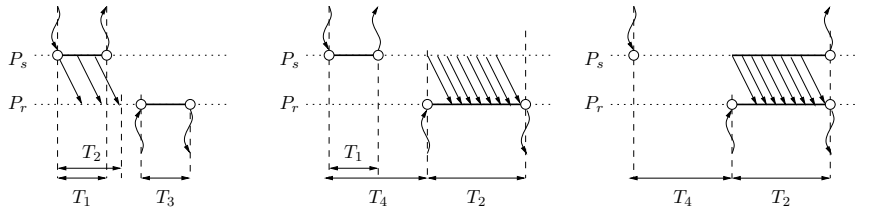
**shadow execution** and to trade off accuracy for simulation speed by benchmarking the execution of program blocks a limited number of time, while skipping these blocks later on and inserting a computation time in the simulated system clock based on the benchmarked value. This corrupts the solution produced by the application, but for data independent applications (those whose behavior does not depend on the results of the computations) this is likely to result in a reasonably accurate execution profile. A related technique, also provided by SMPI, is **memory folding**, whereby multiple simulated processes can share a single copy of the same malloc'd data structure. Again, this can corrupt results and potentially result in inconsistent or illegal data values, but allows larger scale simulations than what would be possible otherwise, and may be reasonable for a large class of parallel applications. These features are disabled by default, and have to be enabled by an expert user by adding annotations to the application source code. Potential areas for future work include improving this process so that it can be semi-automated, and building more sophisticated models based on benchmarked values.

## IV. A "Hybrid" Network Model

In this section, we report some issues that we encountered when comparing the predictions given by existing models to real measurement on a commodity cluster with a hierarchical Ethernet-based interconnection network. The observed discrepancies motivate the definition of a new hybrid model building upon LogGPS and fluid models, that captures all the relevant effects observed during this study. All the presented experiments were conducted on the *graphene* cluster of the Grid'5000 experimental testbed [21], [8]. This cluster comprises 144 2.53GHz Quad-Core Intel Xeon x3440 nodes spread across four cabinets, and interconnected by a hierarchy of 10 Gigabit Ethernet switches (see Figure 2).

### A. Point-to-point communication model

As described previously in Section II, models in the LogP family resort to piece-wise linear functions to account for features such as protocol overhead, switch latency, and the overlap of computation and communication. In the LogGPS model [26] the time spent in the `MPI_Send` and `MPI_Recv` functions is modeled as a **continuous** linear function for small messages ($o + kO_s$ or $o + kO_r$). Unfortunately, as illustrated in Figure 4, this model is unable to account for the full complexity of a real MPI implementation. The measurements presented in Figure 4 were obtained according to the following protocol: To avoid size and sequencing measurement bias,




(a) SMPI Asynchronous mode ($k \leq S_a$)

(b) SMPI Detached mode ($S_a < k \leq S_d$)

(c) SMPI Synchronous mode ($k > S_d$)

| Routine | Condition | Cost |
|---|---|---|
| `MPI_Send` | $k \leq S$ | $T_1$ |
| | $k > S$ | $\max(T_4, 0) + T_2$ |
| `MPI_Recv` | $k \leq S$ | $\min(t_s + T_2 - t_r, 0) + T_3$ |
| | $k > S$ | $\max(-T_4, 0) + T_2$ |

If $k \in I_i$:

$$\begin{cases} T_1 = o_s^{(i)} + kO_s^{(i)} & T_2 = L^{(i)} + k/B^{(i)} \\ T_3 = o_r^{(i)} + kO_r^{(i)} & T_4 = t_r - t_s \end{cases}$$

(d) SMPI communication costs

Figure 3. The "hybrid" network model of SMPI in a nutshell.

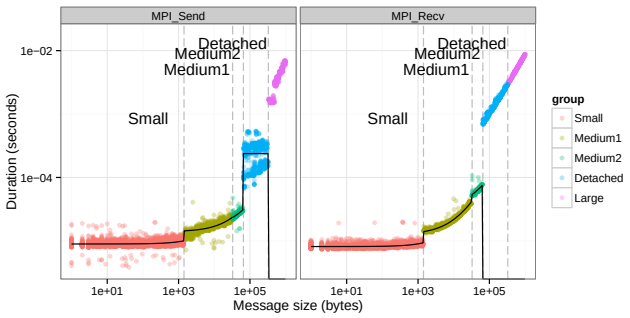



Figure 4. `MPI_Send` and `MPI_Recv` duration as a function of message size.

the message size is exponentially sampled from 1 byte to 100MiB. We ran two "ping" and one "ping-pong" experiments. The ping experiments aim at measuring the time spent in the `MPI_Send` (resp. `MPI_Recv`) function by ensuring that the receiver (resp. sender) is always ready to communicate. The ping-pong experiment allows us to measure the transmission delay. We ran our analysis on the whole set of raw measurements rather than on averaged values for each message size to prevent behavior smoothing and variability information loss. The rationale is to study the asynchronous part of MPI (from the application point of view) without any a priori assumptions on where switching may occur. This approach allows us to clearly identify different modes interpreted as follows:

- **Small** (when $k \leq 1,420$): this mode corresponds to messages that fit in a TCP packet and are sent asynchronously by the kernel. As it induces memory copies, the duration significantly depends on the message size.
- **Medium** (when $1,420 < k \leq 32,768$ or $32,768 < k \leq 65,536 = S_a$): these messages are still sent asynchronously but incur a slight overhead compared to small messages, hence a discontinuity at $k = 1420$. The distinction at $k = 32,768$ does not really correspond to any particular threshold on the sender side but is visible on the receiver side where a small gap is noticed. Accounting for it allows for a better linear fitting accounting for MPI/TCP peculiarities.
- **Detached** (when $65,536 < k \leq 327,680 = S_d$): this mode corresponds to messages that do not block the sender but require the receiver to post the reception before the communication actually takes place.
- **Large** (when $k > 327,680$): for such messages, both sender and receiver synchronize using a rendez-vous protocol before sending data. Except for the waiting time, the durations on the sender side and on the receiver side are very close.

As illustrated by Figure 4, the duration of each mode can be accurately modeled through linear regression. These observations justify the model implemented in SMPI that is described in Figure 3. We distinguish three modes of operation: asynchronous, detached, and synchronous. Each of these modes can be divided in sub-modes when discontinuities are observed. The "ping" measurements are used to instantiate the values of $o_s$, $O_S$, $o_r$, and $O_r$ for small to detached messages. By subtracting $2(o_r + k.O_r)$ from the round trip time measured by the ping-pong experiment, and thanks to a piece-wise linear regression, we can deduce the values of $L$ and $B$. It is interesting to note that similar experiments with `MPI_Isend` and `MPI_Irecv` show that modeling their duration by a constant term $o$ as was done in [26] is not a reasonable assumption neither for simulation nor prediction purposes[1].

While distinguishing these modes may be of little importance when simulating applications that only send particular message sizes, obtaining good predictions in a wide range of settings, and without conducting custom tuning for every simulated application, requires accurately accounting for all such peculiarities. This will be exemplified in Section V.

### B. Topology and contention model

For most network models, dealing with contention comes down to assuming one of the simple **single-port** or **multi-port** models. In the single-port model each node can communicate with only one other node at a time and messages are queued, while in the multi-port model, each node can communicate with every other node simultaneously without any slowdown. Both models oversimplify the reality. Some flow-level models follow a bounded multi-port approach [25], i.e., the communication capacity of a node is limited by the network bandwidth it can exploit, that better reflects the behavior of communications on wide area networks. However, within a cluster or cluster-like environment the mutual interactions between send and receive operations cannot safely be ignored.

[1]More information on how to instantiate the parameters of the SMPI model and about the study of non-blocking operations is available at http://mescal.imag.fr/membres/arnaud.legrand/research/smpi/smpi_loggps.php

To quantify the impact of network contention of a point-to-point communication between two processors in a same cabinet, we artificially create contention and measure the bandwidth as perceived by the sender and the receiver. We place ourselves in the **large** message mode where the highest bandwidth usage is observed and transfer 4 MiB messages.

In a first experiment we increase the number of concurrent transfers from 1 to 19, i.e., half the size of the first cabinet. As the network switch is well dimensioned, this experiment does not create contention: We observe no bandwidth degradation on either the sender side or the receiver side. Our second experiment uses concurrent `MPI_Sendrecv` transfers instead of unidirectional transfers. We increase the number of concurrent bidirectional transfers from 1 to 19 and measure the bandwidth on the sender ($B_s$) and receiver ($B_r$) side. A single-port model, as assumed by LogP-based models, would lead to $B_s + B_r = B$ on average since both directions strictly alternate. A multi-port model, as assumed by other delay models, would estimate that $B_s + B_r = 2 \times B$ since communications would not interfere with each other. However, both fail to model what actually happens, as we observe that $B_s + B_r \approx 1.5 \times B$ on this cluster.

We model this bandwidth sharing effect by enriching the simulated cluster description. Each node is provided with three links: an uplink and a downlink, so that send and receive operations share the available bandwidth separately in each direction; and a specific *limiter* link, whose bandwidth is $1.5 \times B$, shared by all the flows to and from this processor.

Preliminary experiments on other clusters show that this contention parameter seems constant for a given platform, with a value somewhere between 1 and 2. Such value somehow corresponds to the effective limitation due to the card capacity and the protocol overhead. Determining this parameter requires benchmarking each cluster as described in this section. Our set of experiments is available on the previously indicated web page. We are currently working on a more generic benchmarking solution that one could easily use to determine the exact contention factor for any cluster.

This modification is not enough to model contention at the level of the whole *graphene* cluster. As described in Figure 2, it is composed of four cabinets interconnected through 10Gb links. Experiments show that these links become limiting when shared between several concurrent pair-wise communications between cabinets. This effect corresponds to the switch backplane and to the protocol overhead and is captured by describing the interconnection of two cabinets as three distinct links (uplink, downlink, and *limiter* link). The bandwidth of this third link is set to 13 Gb as measured.

The resulting topology is depicted on Figure 5 and is easily described in a compact way within SimGrid. Since SimGrid, on which SMPI is based, is a versatile simulator, incorporating further levels of hierarchy or more complex interconnections if needed would be easy [7].

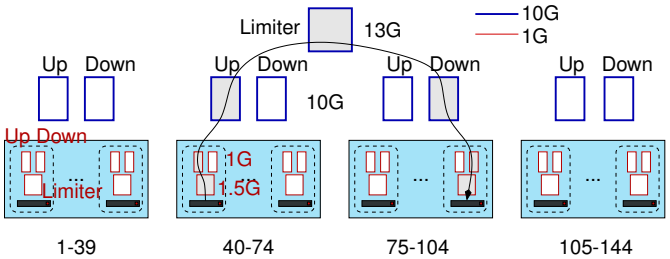


Figure 5. Modeling the graphene cluster: rectangles represent capacity constraints. Grayed rectangles represent constraints involved in a communication from node to node 40 to node 104.

### C. Collective communications model

Many MPI applications spend a significant amount of time in collective communication operations. They are thus crucial to application performance. Several algorithms exist for each collective operation, each of them exhibiting very different performance depending on various parameters such as the network topology, the message size, and the number of communicating processes [19]. A given algorithm can commonly be almost an order of magnitude faster than another in a given setting and yet slower than this same algorithm in another setting. Every widely-used MPI implementation thus provides several algorithms for each collective operation and carefully selects the best one at runtime. For instance, OpenMPI provides a dozen distinct algorithms for the `MPI_Alltoall` function, and the code to select the right algorithm for a given setting is several thousand lines long. Note that the selection logic of the various MPI implementations is highly dependent on the implementation and generally embedded deep within the source code.

Our [in]validation experiments quickly highlighted the importance of adhering as closely as possible to this logic. Hence SMPI now implements **all** the collective algorithms and selection logics of both OpenMPI and MPICH and even a few other collective algorithms from Star MPI [19]. Deciding which selector and which algorithms are used can be specified from command line, which allows users to test within simulation whether replacing a default algorithm by another may help or not on a particular combination of platform/application.

## V. Model [In]Validation Experiments

### A. Methodology

All the experiments presented hereafter have been done on the *graphene* cluster that was described in the previous section. The studied MPI applications were compiled and linked using OpenMPI 1.6. For comparison with simulated executions purposes, we instrumented these applications with TAU [38]. The simulated executions have been performed either off-line or on-line as SMPI supports both modes. The file describing the simulated version of the *graphene* cluster was instantiated with values obtained independently from the studied applications. We used the techniques detailed in the previous section to obtain these values. In what follows we compare execution times measured on the *graphene* cluster to

simulated times obtained with the *hybrid* model proposed in Section IV, the *LogGPS* model that supersedes all the delay-based models, and a *fluid* model that is a basic linear flow-level model whose validation for WAN studies was done in [43].

We did not limit our study to overall execution times as they may hide compensation effects, and do not provide any information as to whether an application is compute or communication bound or how different phases may or may not overlap. Our experimental study makes use of Gantt charts to compare traces visually as well as quantitatively. We rely on CSV files, R, and org-mode to describe the complete workflow going from raw data the graphs presented in this paper, ensuring full reproducibility of our analysis.

### B. NAS Parallel Benchmarks

The NAS Parallel Benchmarks (NPB) are a suite of programs commonly used to assess the performance of parallel platforms. In this article, we only report results for two of these applications but results for other applications are also available in [6]. The LU benchmark is an iterative computation of a LU factorization. At each iteration, it exhibits a communication scheme that forms a wave going from the first process to the last one and back. The second studied benchmark is CG (Conjugate Gradient). It has a complex communication scheme that is composed of a large number of point-to-point transfers of small messages. Moreover, processors are organized in a hierarchy of groups. At each level communications occur within a group and then between groups. This benchmark is then very sensitive to the mapping of the MPI processes on to the physical processors with regard to network organization, particularly in non-homogeneous topologies. For this series of experiments, we use the off-line simulation capacities of SMPI, i.e., execution traces of the benchmark are first acquired from a real system and then replayed in a simulated context.

Both benchmarks are evaluated with two class of instances, B, and C, where C is the larger instance. A classical goal when studying the performance of an application or of a new cluster is to evaluate how well the application scales for a given instance. Figure 6 shows the speedup as measured on the *graphene* cluster and as obtained with the studied models. For every experiment we ensured that both simulated and real node mapping corresponds to each others.

For the LU benchmark we can see that for the class B instances, the *hybrid* and *LogGPS* model provide an excellent estimation of speedup evolution. As could be expected, the *fluid* model provides an over-optimistic estimation, which can be explained by its poor ability to accurately model transmission time of small messages and computation/communication overlap. For the class C instance, although the *hybrid* model provides a slightly better estimation than the *LogGPS* model, both predict an optimistic scaling of this application. We think, this can be explained by the fact that none of these models include a noise component. Indeed, the communication pattern of LU is very sensitive to noise as each process has to wait for the reception of a message before sending its own data. Such a phenomenon is given by Figure 7 that shows a period

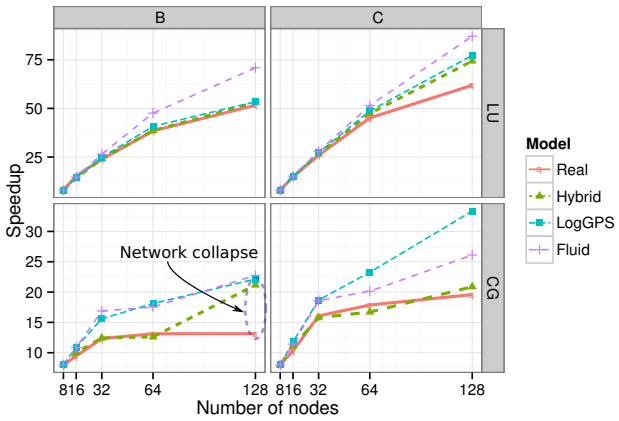



Figure 6. Comparison between simulated and actual execution times for three NAS parallel benchmarks.

of 0.2 seconds of the execution of the LU benchmark with 32 processes. The upper part of this figure displays the actual execution while the lower corresponds to the simulation of the same phase with the *hybrid* model. Deterministic models such as *LogGPS* or *hybrid* slightly underestimate these synchronization phases and small delay adds up, which hinders the application scalability. However, Figure 7 clearly shows that despite the small time scale, the simulation correctly renders the general communication pattern of this benchmark and that Figure 6 does not hide a bad, but lucky, estimation of each component of the execution time.

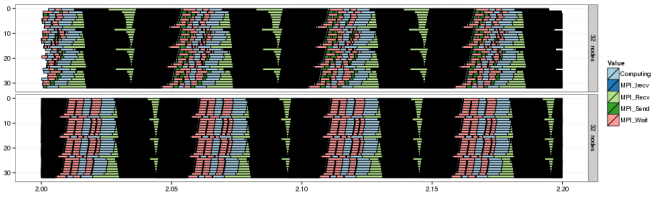


Figure 7. Part of the Gantt chart of the execution of the LU benchmark with 32 processes. The upper part displays the actual execution while the lower is the simulation with the *hybrid* model. Black areas comprise thousands wave-structured micro messages.

The results obtained for the CG benchmark are more discriminating. Indeed, this benchmark transfers messages that never fall in the *large* category. As can be observed, both the *LogGPS* and *fluid* models fail to provide good estimations and largely underestimate the total time. Our *hybrid* model produces excellent estimation except on the class B instance with 128 nodes. To determine if the source of this gap comes from a bad estimation by the model or a problem during the actual execution, we compared more carefully the two traces. Our main suspect was the actual execution, which is confirmed by the Gantt chart presented in Figure 8.

The execution time on the class B instance is 14.4 seconds while the prediction of the *hybrid* model is only of 9.9 seconds. We can see two outstanding zones of `MPI_Send` and `MPI_Wait`. Such operations typically take few microseconds

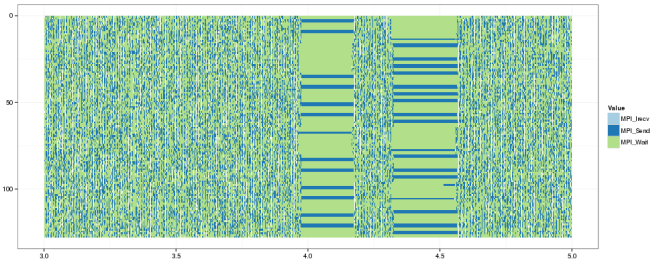


Figure 8. Two seconds Gantt-chart of the real execution of a class B instance of CG for 128 process.

to less than a millisecond. Here they take 0.2 seconds. Our guess is that, due to a high congestion, the switch drops packets and slows down one (or several) process to the point where it stops sending until a timeout of .2 seconds is reached. Because of the communication pattern, blocking one process impacts all the other processes. This phenomenon occurs 24 times leading to a delay of 4.86 seconds. Without this bad behavior, the real execution would take 9.55 seconds, which is extremely close to the 9.85 seconds prediction of the *hybrid* model and would allow the effective speedup to match perfectly its prediction. The same phenomenon is also present for class C although it is less noticeable. In both cases, the speedup shape predicted by the *hybrid* model is non-trivial since it comprises a plateau from 32 to 64 nodes. Such shape can actually be well explained by the hierarchical structure of its communication pattern and how it maps to the network topology. The *LogGPS* model fails at modeling such aspects and would predict an excellent scaling.

Although we do not know for sure yet, we think the timeout issues we encountered could be somehow similar to what is known as the *TCP incast problem* and which has been observed in cloud environments [11]. Such delays are linked to the default TCP re-transmission timeout, which is equal to .2s by default in Linux. Although such value has recently been decreased from 1s to .2s to adapt with recent evolution of Internet characteristics, it remains inadequate for a cluster. Such protocol collapse would clearly need to be fixed in a production environment and we are currently investigating whether decreasing the TCP re-transmission timeout to a drastically smaller value than .2 seconds would help or not as it is not clear that HPC variants of TCP would really help solving such issue.

### C. Collective Communications

The NAS parallel benchmarks do not heavily rely on `MPI` collective communication primitives but instead implement a static communication pattern. We conduct the (in)validation with the study of an isolated MPI collective operation at a time. On medium size clusters, simple operations like broadcast only incur minor network contention toward the end of the operation and may thus be correctly predicted with simple models like *LogGPS*. Therefore, although we conducted similar studies for most commonly used operations, we only report the results for the `MPI_Alltoall` function as it is the most likely to be impacted by network contention. Here we aim at assessing the validity of contention modeling as the message size varies rather than the impact of using different algorithms. To this end, although we extended SMPI so that it implements **all** the collective algorithms and selection logics of both OpenMPI and MPICH, we enforce the use of a single algorithm, i.e., the pairwise-exchange algorithm [39], for all message sizes. We ran the experiment five times in a row and only kept the best execution time for each message size. Indeed, we noticed that the first communication is always significantly slower than the subsequent ones, which tends to indicate that TCP requires some warm-up time. This phenomenon has also been noticed in experiments assessing the validity of the BigSIM simulation toolkit, where the same workaround was applied.

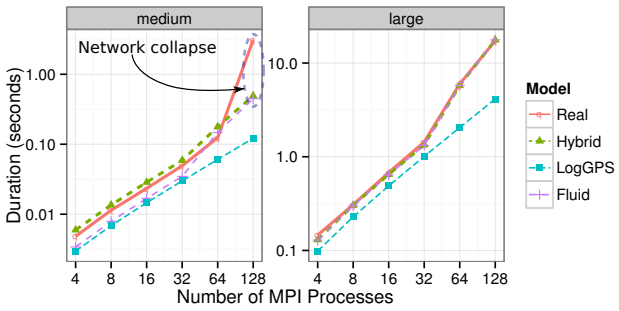



Figure 9. Comparison between simulated and actual execution times for the `MPI_Alltoall` operation.

Figure 9 shows (in logarithmic scale) the evolution of simulated and actual execution times for the `MPI_Alltoall` operation when the number of MPI processes varies for the three network models. Results are presented for two message sizes: *medium* messages of 100 kiB and *large* messages of 4 MiB. We can see that for large messages, the *hybrid* and *fluid* models both achieve excellent predictions (within 10%). Unsurprisingly, the *LogGPS* model is overly optimistic in such setting and completely underestimates the effects of network contention. Its prediction error can be **up to a factor of 4** when 128 processors are involved in the All-to-All operation.

For medium messages, the *hybrid* model is again the best contender, with a low prediction error for up to 64 nodes, while the *LogGPS* model is again too optimistic. For such a message size, the lack of latency and bandwidth correction factors in the *fluid* model leads to a clear underestimation of the execution time. Interestingly, when 128 processes are involved in the collective communication, the actual execution time increases dramatically while simulated times continue to follow the same trend. The reason for such a large increase can again be explained by massive packet dropping in the main switch that leads to timeouts, and unexpected re-emissions, hence incurring significant delays compared to usual transmission time. Modeling such phenomenon would probably be quite difficult and of little interest since fixing this problem on the real platform would be much more useful.

## VI. Simulating a Real Application

Developing a research prototype that allows for simulation of a few benchmarks already requires a lot of efforts and is useful to demonstrate the effectiveness of an idea. However, it does not allow others to build upon it. Then, we also ensured SMPI could also be used to simulate complex real applications such as the full LinPACK suite [18], Sweep3D [4], or BigDFT, which is an open-source Density Functional Theory (DFT) massively parallel electronic structure code [20], and the geodynamics application SpecFEM3D [34], which is part of the PRACE benchmark. SMPI also is tested upon 80% of the MPICH2 test suite and against a large subset of the MPICH3 test suite every night. We can thus claim that SMPI is not limited to toy applications but can effectively be used for the analysis of real scientific applications.

In this section we aim at demonstrating the capacity of SMPI to simulate a real, large, and complex MPI application. To this end, we use BigDFT, which is the sole electronic structure code based on systematic basis sets which can use hybrid supercomputers and is able to scale particularly well (95% of efficiency with 4096 nodes on Curie [14]). This is why it is one of the eleven real scientific applications that have been selected in the Mont-blanc project [30] to assess the potential of low-power embedded components based clusters to address future Exascale HPC needs. One of the objectives of the Mont-Blanc project is thus to develop prototypes of HPC clusters using low power commercially available embedded technology such as ARM processors and Ethernet technologies.

The first Mont-Blanc prototype is expected to be available during the year 2014. It will be using Samsung Exynos 5 Dual Cortex A15 processors with an embedded Mali T604 GPU and will be using Ethernet for communication. In order to start evaluating the applications before 2014, a small cluster of ARM system on chip was built. It is named Tibidabo and is hosted at the Barcelona Supercomputing Center. Tibidabo [36] is an experimental HPC cluster built using NVIDIA Tegra2 chips, each a dual-core ARM Cortex-A9 processor. The PCI Express support of Tegra2 is used to connect a 1Gb Ethernet NIC, and the board are interconnected hierarchically using 48-port 1 GbE switches. The results we present in this section have been obtained using this platform.

For our experiments, we disable the OpenMP and GPU extensions at compile time to study behaviors related to MPI operations. BigDFT alternates between regular computation bursts and important collective communications. Moreover the set of collective operations that is used may completely change depending on the instance, hence the need to use online simulation. In the following experiments, we used MPICH 3.0.4 [39] and Extrae [5] for runtime incompatibility issues between OpenMPI, Tau and BigDFT. Last, while this application can be simulated by SMPI without any modification to the source code, its large memory footprint means that running the simulation on a single machine would require an improbably large amount of RAM. Applying the memory folding and shadow execution techniques mentioned in Section III and detailed in [12], we were able to simulate the execution of BigDFT with 128 processes, whose peak memory footprint is estimated to 71 GiB, on a single laptop using less than 2.5GB of memory. Such memory requirement could be further improved with additional manual annotations but it was sufficient for our needs.

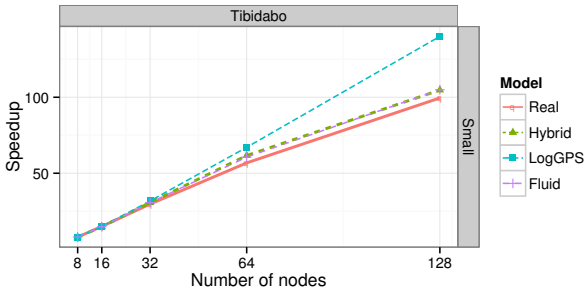



Figure 10. Evaluating scalability of BigDFT on Tibidabo both through real executions and simulation. The LogGPS model fails to model the slowdown incurred by the hierarchical and irregular network topology.

Figure 10 shows the comparison of the speedup evolution as measured on the *tibidabo* cluster on a small instance. This instance has a relatively low communication to computation ratio (around 20% of time is spent communicating when using 128 nodes and the main used operations are `MPI_Alltoall`, `MPI_Alltoallv`, `MPI_Allgather`, `MPI_Allgatherv` and `MPI_Allreduce`) despite the slow computations of tibidabo. This instance is thus particularly difficult to model and is expected to have a limited scalability, which one may want to observe in simulation first to avoid wasting resources or to assess the relevance of upgrading hardware. As expected, the *LogGPS* model predicts an over-optimistic perfect scaling whereas both the *fluid* and the *hybrid* models succeed in accounting for the slowdown incurred by the hierarchical and irregular network topology of this prototype platform. We think this kind of observation really questions the use of the *LogGPS* model for scalability studies.

To further demonstrate the usability of our tool, we want to mention that simulating 64 nodes of tibidabo, which is made of relatively slow ARM processors, on a Xeon 7460 with partial on-line simulation takes twice as less time (10 minutes) than running the code for real (20 minutes).

## VII. Related Work

Packet-level network simulations are usually implemented as discrete-event simulations with events for packet emission or reception as well as network protocol events. Such simulations reproduce the real-world communication behavior down to movements of individual packets. Some tools following this approach, e.g., NS2, NS3, or OMNet++, have been widely used to design network protocols or understand the consequences of protocol modifications [29]. However, such fine-grain network models are difficult to instantiate with realistic parameter values for large-scale networks and generally suffer from scalability issues. While parallel discrete-event simulation techniques [47], [31] may speed up such simulations, the possible improvements remain quite limited.

Some projects model MPI collective communications with simple analytic formulas [2], [40]. This allows for quick estimations and may provide a reasonable approximation for simple and regular collective operations, but is unlikely to accurately model the complex optimized versions that can be found in most MPI implementations. An orthogonal approach is to thoroughly benchmark collective operations to measure the distribution of communication times with regard to the message size and number of concurrent flows. Then, these distributions are used to model the interconnection network as a black box [22]. This approach has several drawbacks. First, it does not accurately model communication/computation overlap. Second, it cannot take the independence of some concurrent communications into account. Third, it provides little to no information on how the performance of collective operations could be improved. Finally, it does not allow for performance extrapolation on a larger machine with similar characteristics and it provides little insight into the causes of poor performance. A third approach consists in tracing the execution of collective operations and then replaying the obtained trace using one of the aforementioned delay models [24], [47], [2], [40]. However, most tools [24], [40], [2], [16] use a very basic network topology model that does not account for the complexity of modern platforms. Furthermore, current implementations of the MPI standard dynamically select from up to a dozen different communication algorithms when executing a collective operation depending on message size and on the number of involved nodes. Thus, using the right algorithm becomes critical when trying to predict performance.

Studying the behavior of complex HPC applications or operations and characterizing HPC platforms through simulation has been at the heart of many research projects and tools for decades. Such tools differ by their capabilities, their structure, and by the network models they implement. BigSIM [47], LAPSE [16], MPI-SIM [3], or the work in [37] rely on simple delay models (affine point-to-point communication delay based on link latency and bandwidth). Other tools, such as Dimemas [2], LogGOPSim [24], or PHANTOM [46] use variants of the LogP model to simulate communications. Note that BigSIM also offers an alternate simulation mode based on a complex and slow packet-level simulator. This approach is also followed by MPI-NeTSim [33] that relies on OMNeT++. Finally PSINS [40] and PEVPM [22] provide complex custom models derived from intensive benchmarking to model network contention.

## VIII. Conclusion and Future Work

In this paper, we demonstrated that accurate modeling and performance prediction for a wide range of parallel applications requires proper consideration of many aspects of the underlying communication architecture, including the breakdown of collective communications into their component point-to-point messages, the interconnect topology, and contention between competing messages that are sent simultaneously over the same link. Even relatively minor inaccuracies may compromise the soundness of the simulation, yet none of the models previously used in the literature give due consideration to these factors. We described the implementation of a proposed *hybrid* network model that improves on this situation within SMPI, and shown that SMPI-based simulations do a better job of tracking real-world behavior than those implemented in competing simulation toolkits.

Our priority in this work was the validation of the model at a small scale and for TCP over Ethernet networks. However, we believe SMPI will prove very useful to application developers by allowing them to debug parallel applications or to study the impact of selecting different collective communication algorithms without wasting resources of a cluster. It also provides a good comparison point that helps knowing whether the platform or the application behaves as expected or not. Previous models were expected to provide over optimistic evaluations and were thus of little use. Finally, we think this tool will reveal very useful for efforts such as the European Mont-Blanc project [30], [36], which aims at prototyping exascale platforms using low-power embedded processors interconnected by Ethernet. A next step will be to analyze its adequacy for simulating larger platforms when we can get access to such large machines for experimental purposes. The study should then extend to other kinds of interconnects (such as InfiniBand) and more complicated topologies (e.g., torus or fat trees).

As a base line, we advocate for an open-science approach, which should enable other scientists to reproduce the experiments done in this paper. For that purpose, the traces and scripts used to produce our analysis are available [45]. Accordingly, SMPI and all the software stack are provided as open-source software available for download from the SimGrid website: http://simgrid.gforge.inria.fr/.

## IX. Acknowledgments

This work is partially supported by the SONGS ANR project (11-ANR-INFRA-13) and the CNRS PICS N° 5473. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

## References

[1] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: Incorporating Long Messages Into the LogP Model – One Step Closer Towards a Realistic Model for Parallel Computation. In *Proc. of the 7th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 95–105, Santa Barbara, CA, 1995.

[2] R. M. Badia, J. Labarta, J. Giménez, and F. Escalé. Dimemas: Predicting MPI Applications Behaviour in Grid Environments. In *Proc. of the Workshop on Grid Applications and Programming Tools*, June 2003.

[3] R. Bagrodia, E. Deelman, and T. Phan. Parallel Simulation of Large-Scale Parallel Applications. *International Journal of High Performance Computing and Applications*, 15(1):3–12, 2001.

[4] R. S. Baker and K. R. Koch. An $s_n$ algorithm for the massively parallel CM-200 computer. *Nuclear Science and Engineering*, 128(3):312–320, Mar. 1998. Available at http://wwwc3.lanl.gov/pal/software/sweep3d/.

[5] Barcelona Supercomputer Center. Extrae. http://www.bsc.es/computer-sciences/extrae/.

[6] P. Bedaride, S. Genaud, A. Degomme, A. Legrand, G. Markomanolis, M. Quinson, L. Stillwell, Mark, F. Suter, and B. Videau. Improving Simulations of MPI Applications Using A Hybrid Network Model with Topology and Contention Support. Rapport de recherche RR-8300, INRIA, May 2013.

[7] L. Bobelin, A. Legrand, D. A. G. Márquez, P. Navarro, M. Quinson, F. Suter, and C. Thiery. Scalable Multi-Purpose Network Representation for Large Scale Distributed System Simulation. In *Proc. of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 220–227, Ottawa, Canada, May 2012.

[8] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lantéri, J. Leduc, N. Melab, G. M. andR. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, Nov. 2006.

[9] L. Carrington, M. Laurenzano, and A. Tiwari. Inferring large-scale computation behavior via trace extrapolation. In *Large-Scale Parallel Processing workshop (IPDPS'13)*, 2013.

[10] H. Casanova, A. Legrand, and M. Quinson. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *Proc. of the 10th IEEE International Conference on Computer Modeling and Simulation*, Cambridge, UK, Mar. 2008.

[11] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. Understanding tcp incast throughput collapse in datacenter networks. In *Proc. of the 1st ACM workshop on Research on enterprise networking*, WREN '09, pages 73–82. ACM, 2009.

[12] P.-N. Clauss, M. Stillwell, S. Genaud, F. Suter, H. Casanova, and M. Quinson. Single Node On-Line Simulation of MPI Applications with SMPI. In *Proc. of the 25th IEEE Intl. Parallel and Distributed Processing Symposium (IPDPS)*, Anchorage, AK, May 2011.

[13] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Proc. of the fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP)*, pages 1–12, San Diego, CA, 1993.

[14] The curie supercomputer. http://www-hpc.cea.fr/en/complexe/tgcc-curie.htm.

[15] F. Desprez, G. S. Markomanolis, and F. Suter. Improving the Accuracy and Efficiency of Time-Independent Trace Replay. In *Proc. of the 3rd International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, Salt Lake City, UT, Nov. 2012.

[16] P. Dickens, P. Heidelberger, and D. Nicol. Parallelized Direct Execution Simulation of Message-Passing Parallel Programs. *IEEE Transactions on Parallel and Distributed Systems*, 7(10):1090–1105, 1996.

[17] B. Donassolo, H. Casanova, A. Legrand, and P. Velho. Fast and Scalable Simulation of Volunteer Computing Systems Using SimGrid. In *Proc. of the Workshop on Large-Scale System and Application Performance (LSAP)*, Chicago, IL, June 2010.

[18] J. J. Dongarra, P. Luszczek, and A. Petitet. The linpack benchmark: Past, present, and future. concurrency and computation: Practice and experience. *Concurrency and Computation: Practice and Experience*, 15:2003, 2003.

[19] A. Faraj, X. Yuan, and D. Lowenthal. STAR-MPI: self tuned adaptive routines for MPI collective operations. In *Proc. of the 20th annual international conference on Supercomputing*, ICS '06, pages 199–208, New York, NY, USA, 2006. ACM.

[20] L. Genovese, A. Neelov, S. Goedecker, T. Deutsch, S. A. Ghasemi, A. Willand, D. Caliste, O. Zilberberg, M. Rayson, A. Bergman, and R. Schneider. Daubechies Wavelets as a Basis Set for Density Functional Pseudopotential Calculations. *Journal of Chemical Physics*, 129(014109), 2008.

[21] Technical specification of the network interconnect in the graphene cluster of grid'5000. https://www.grid5000.fr/mediawiki/index.php/Nancy:Network.

[22] D. A. Grove and P. D. Coddington. Communication benchmarking and performance modelling of mpi programs on cluster computers. *Journal of Supercomputing*, 34(2):201–217, Nov. 2005.

[23] M.-A. Hermanns, M. Geimer, F. Wolf, and B. Wylie. Verifying Causality between Distant Performance Phenomena in Large-Scale MPI Applications. In *Proc. of the 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pages 78–84, Weimar, Germany, Feb. 2009.

[24] T. Hoefler, C. Siebert, and A. Lumsdaine. LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model. In *Proc. of the ACM Workshop on Large-Scale System and Application Performance*, pages 597–604, Chicago, IL, June 2010.

[25] B. Hong and V. K. Prasanna. Adaptive Allocation of Independent Tasks to Maximize Throughput. *IEEE Transactions on Parallel and Distributed Systems*, 18(10):1420–1435, Oct. 2007.

[26] F. Ino, N. Fujimoto, and K. Hagihara. LogGPS: a Parallel Computational Model for Synchronization Analysis. In *Proc. of the eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming (PPoPP)*, pages 133–142, Snowbird, UT, 2001.

[27] T. Kielmann, H. E. Bal, and K. Verstoep. Fast Measurement of LogP Parameters for Message Passing Platforms. In *Proc. of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, IPDPS '00, pages 1176–1183, London, UK, UK, 2000. Springer-Verlag.

[28] G. F. Lucio, M. Paredes-farrera, E. Jammeh, M. Fleury, and M. J. Reed. Opnet modeler and ns-2: Comparing the accuracy of network simulators for packet-level analysis using a network testbed. In *Proc. of the 3rd WEAS International Conference on Simulation, Modelling and Optimization (ICOSMO*, pages 700–707, 2003.

[29] C. Minkenberg and G. Rodriguez. Trace-Driven Co-Simulation of High-Performance Computing Systems Using OMNeT++. In *Proc. of the 2nd International Conference on Simulation Tools and Techniques (SimuTools)*, Rome, Italy, 2009.

[30] Mont-Blanc: European Approach Towards Energy Efficient High Performance. Montblanc. http://www.montblanc-project.eu/.

[31] M. Mubarak, C. D. Carothers, R. Ross, and P. Carns. Modeling a million-node dragonfly network using massively parallel discrete-event simulation. *High Performance Computing, Networking Storage and Analysis, SC Companion:*, 0:366–376, 2012.

[32] A. Núñez, J. Fernández, J.-D. Garcia, F. Garcia, and J. Carretero. New Techniques for Simulating High Performance MPI Applications on Large Storage Networks. *Journal of Supercomputing*, 51(1):40–57, 2010.

[33] B. Penoff, A. Wagner, M. Tüxen, and I. Rüngeler. MPI-NeTSim: A network simulation module for MPI. In *Proc. of the 15th IEEE Intl. Conference on Parallel and Distributed Systems*, Shenzen, China, Dec. 2009.

[34] D. Peter, D. Komatitsch, Y. Luo, R. Martin, N. Le Goff, E. Casarotti, P. Le Loher, F. Magnoni, Q. Liu, C. Blitz, T. Nissen-Meyer, P. Basini, and J. Tromp. Forward and Adjoint Simulations of Seismic Wave Propagation on Fully Unstructured Hexahedral Meshes. *Geophysical Journal International*, 186(2):721–739, 2011.

[35] M. Quinson, C. Rosa, and C. Thiéry. Parallel simulation of peer-to-peer systems. In *Proceedings of the 12th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'12)*. IEEE Computer Society Press, may 2012.

[36] N. Rajovic, N. Puzovic, L. Vilanova, C. Villavieja, and A. Ramirez. The low-power architecture approach towards exascale computing. In *Proc. of the second workshop on Scalable algorithms for large-scale systems*, ScalA '11. ACM, 2011.

[37] R. Riesen. A Hybrid MPI Simulator. In *Proc. of the IEEE International Conference on Cluster Computing*, Barcelona, Spain, Sept. 2006.

[38] S. Shende and A. D. Malony. The Tau Parallel Performance System. *International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[39] R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of collective communication operations in MPICH. *International Journal of High Performance Computer Applications*, 19(1):49–66, 2005.

[40] M. Tikir, M. Laurenzano, L. Carrington, and A. Snavely. PSINS: An Open Source Event Tracer and Execution Simulator for MPI Applications. In *Proc. of the 15th International EuroPar Conference*, volume 5704 of *Lecture Notes in Computer Science*, pages 135–148, Delft, Netherlands, Aug. 2009.

[41] Top 500 supercomputer sites. http://top500.org.

[42] P. Velho, L. Schnorr, H. Casanova, and A. Legrand. Flow-level network models: have we reached the limits? Rapport de recherche RR-7821, INRIA, Nov. 2011.

[43] P. Velho, L. Schnorr, H. Casanova, and A. Legrand. On the validity of flow-level tcp network models for grid and cloud simulations. *ACM: Transactions on Modeling and Computer Simulation*, 2013. Accepted for publication. See [42] for a preliminary version.

[44] X. Wu and F. Mueller. ScalaExtrap: trace-based communication extrapolation for SPMD programs. In *Proc. of the 16th ACM symposium*

*on Principles and Practice of Parallel Programming (PPoPP'11)*, pages 113–122, 2011.

[45] Improving simulations of MPI applications using a hybrid network model with topology and contention support. http://mescal.imag.fr/membres/arnaud.legrand/research/smpi/smpi_sc13.php. Online version of the Article with access to the experimental data and scripts (access org source at the bottom of the document).

[46] J. Zhai, W. Chen, and W. Zheng. PHANTOM: Predicting Performance of Parallel Applications on Large-Scale Parallel Machines Using a Single Node. In *Proc. of the 15th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, pages 305–314, Jan. 2010.

[47] G. Zheng, G. Kakulapati, and L. Kale. BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines. In *Proc. of the 18th International Parallel and Distributed Processing Symposium*, Santa Fe, NM, Apr. 2004.