Improving Simulations of MPI Applications Using A Hybrid Network Model with Topology and Contention Support

Paul Bédaride
Loria/INRIA/Univ. of Nancy
Nancy, France
Paul.Bedaride@loria.fr

Augustin Degomme
CNRS/INRIA/Univ. of Grenoble
Grenoble, France
Augustin.Degomme@imag.fr

Stéphane Genaud
University of Strasbourg
Strasbourg, France
genaud@unistra.fr

Arnaud Legrand
CNRS/INRIA/Univ. of Grenoble
Grenoble, France
Arnaud.Legrand@imag.fr

George S. Markomanolis
INRIA, LIP, ENS Lyon
Lyon, France
gmarko01@ens-lyon.fr

Martin Quinson
Loria/INRIA/Univ. of Nancy
Nancy, France
Martin.Quinson@loria.fr

Mark Stillwell
Engineering Computing
Cranfield University, U.K.
m.stillwell@cranfield.ac.uk

Frédéric Suter
IN2P3 Computing Center, CNRS
Lyon-Villeurbanne, France
fsuter@cc.in2p3.fr

Brice Videau
CNRS/Univ. of Grenoble
Grenoble, France
Brice.Videau@imag.fr

ABSTRACT

Proper modeling of collective communications is essential for understanding the behavior of medium-to-large scale parallel applications, and even minor deviations in implementation can adversely affect the prediction of real-world performance. We propose a hybrid network model extending LogP based approaches to account for topology and contention in high-speed TCP networks. This model is validated within SMPI, an MPI implementation provided by the SimGrid simulation toolkit. With SMPI, standard MPI applications can be compiled and run in a simulated network environment, and traces can be captured without incurring errors from tracing overheads or poor clock synchronization as in physical experiments. SMPI provides features for simulating applications that require large amounts of time or resources, including selective execution, ram folding, and off-line replay of execution traces. We validate our model by comparing traces produced by SMPI with those from other simulation platforms, as well as real world environments.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Performance, Analysis & Tools, Programming Systems

Keywords

Simulation for performance, Software engineering

1. INTRODUCTION

Correctly modeling collective communications, including their low-level breakdown into individual point-to-point messages and how these messages interact and interfere with one another within the network environment, is essential for understanding the higher-level behavior of medium and large scale parallel applications. 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 LogP based approaches to better account for topology and contention in high-speed TCP networks. This is part of a larger effort to study distributed computing platforms as natural objects, recognizing that the remarkable complexity of such systems makes it increasingly difficult to make useful claims from a purely theoretical viewpoint. Indeed, the computation and communication behavior of many, if not most, existing scientific codes is understood at only the most basic level, even by the application programmers. We propose to study these systems through simulation based on empirically validated models, thus turning the tools and techniques afforded by advanced computational resources back on those resources themselves.

This approach is further justified by the recent recognition of computational science and simulation as an essential "third pillar" of scientific methodology, on par with and distinct from experimentation and theory. Well thought out simulations can provide results that are more easily understood and analyzed than a physical experiment at significantly lower costs. Further, because of the difficulty in ensuring the precise configuration of the experimental environment, in some fields the results from simulation are regarded as more accurate and informative than those from physical experiments; A notable example of this can be seen in the field of fluid mechanics, where the overall importance of micro-bubbles in reducing drag coefficients for moving objects (such as ships)

had been long understood, but difficulties in observational methods made it impossible to make significant advances prior to the advent of techniques for simulation [29]. The real-world environment for distributed systems is plagued with similar problems, in that proper instrumentation and monitoring is extremely difficult and may alter the system behavior. Real-world networks are also subject to transient or periodic faults caused by unknown and hard-to-detect hardware problems that can invalidate the applicability of results outside of the testing facility. Another advantage of simulation is that it makes it easier for independent third parties to analyze experiments and reproduce results, thus forwarding the cause of open science.

In this context, network modeling is a critical point. Tools that address such concerns tend to rely either on "accurate" packet-level simulations or on simple analytical models. Packet-level simulation has high overheads, resulting in simulations that may take significantly longer to run than the corresponding physical experiments. Additionally, while packet-level simulation has long been considered the "gold standard" for modeling the behavior of network communications, this assumption has not been rigorously validated and it may be the case large disparities exist between predicted and actual behavior. Recent work suggests that well-tuned flow-based simulation may be able to provide more accurate results at lower costs [28]. The problem with analytic approaches is that many simulators implement communications from the ground up, using simplified models that ignore phenomena such as network topology and contention or even how the collective communications are actually implemented. While these inaccuracies have been long-acknowledged within the field, until recently there were few alternatives, and much work has been done with the intuition that while these simplifying assumptions might make it difficult to predict exact performance on a particular platform, overall application behavior would remain basically unchanged. Our own experiments have shown that this is simply not the case.

We seek to improve the accuracy of simulations of medium-to-large scale parallel distributed applications, while capturing 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. The proposed model is validated within SMPI, an implementation of the Message Passing Interface (MPI) that interfaces directly with the SimGrid simulation toolkit [5]. 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 also provides a number of useful features for predicting the behavior of applications that may require large amounts of time or system resources to run, including selective execution, ram folding, and off-line simulation by replay of execution traces. We validate our results by comparing application traces produced by SMPI using our network model with those from real world environments. Further details regarding SMPI, including comparisons with other simulation platforms are given in Section 3.

In this article, we explain our efforts in proposing more realistic network models than what was proposed in other tools and how it improves upon their ability to derive more meaningful results. We justify our results by comparing traces produced by SMPI with those from other simulation platforms, as well as traces gathered from physical experiments on real-world platforms.

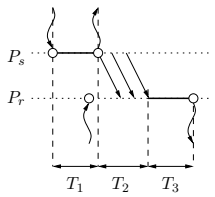
More specifically, our contributions are:

- 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 this model in detail and make comparisons to other models proposed within the literature.
- 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.
- 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 provide comparisons with competing simulation frameworks.
- We demonstrate the effectiveness on our proposal by making a thorough study of the validity of our models against hierarchical clusters using Gigabit Ethernet with TCP as it is known to be much more difficult than high-end HPC networks that are more regular.

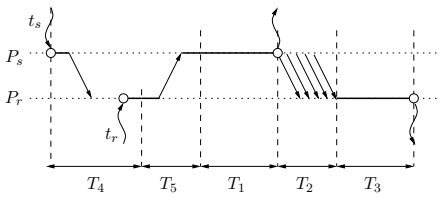
This paper is organized as follows: In Section 2 we discuss related work, particularly in the area of network modeling. In Section 3 we describe SMPI, our chosen implementation, along with some key features that make it suitable for simulation of large scale systems, and comparisons to competing simulation platforms. In Section 4 we describe our proposed hybrid network model, focusing particularly how this model captures the complexities of network topology and message contention. In Section 5 we describe experiments conducted to validate the model, including comparisons of traces produced using other approaches as well as from real-world environments. In Section 6 we demonstrate the capacity of SMPI to simulate complex MPI applications on a single machine. We conclude in Section 7 and also provide a description of proposed areas of future work.

2. RELATED WORK

In the High Performance Computing (HPC) field, accurately predicting the execution time of parallel applications is of uttermost importance to assess their scalability. 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. Such optimizations generally result in complex **heterogeneous performance**. Also, HPC platforms usually rely on a complex network interconnection



(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 [16] in a nutshell.

topology, including switch hierarchies or mesh-based arrangements where connections between nodes map onto abstract N-dimensional figures such as toruses or hypercubes. These arrangements lead to non-homogeneous communication capabilities between processors and possible bottlenecks depending on the application workload. Another important factor is the **network protocol** provided by interconnect fabric. InfiniBand has very stable and regular transmission times, but a simple congestion control mechanism. The result is that well-provisioned platforms suffer from almost no congestion, but cheaper settings may experience terrible performance degradation in the case of network overload. Conversely, Gigabit Ethernet networks using TCP-like protocols exhibit much more variable performance, but are expected to be more scalable due to better handling of network congestion. Finally, the MPI standard includes a very rich set of **collective operations** that have been optimized over time and are still the objects of active research. Performance may vary by several orders of magnitude between "good" and "bad" implementations of a given collective operation. In what follows we describe a number of network modeling approaches and how they address these various issues.

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, which will contain both user data and control information, such as flow-control digits, or flits. 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 [18]. 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 [32, 20] may speed up such simulations, the possible improvements remain quite limited as the underlying systems are usually not well separable. That is, data-interdependencies necessitate the use of shared data structures, which cut down on the efficiency of parallel simulation.

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 [7, 1, 17, 16]. The LogP model was originally proposed by Culler et al. [7] 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. 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 for short messages; 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, as the sender is generally released before the message is actually received on the receiver side. While this model somehow reflects the way specialized HPC networks used to work in the early 90s, it ignores many important aspects.

The LogGP model proposed in [1] introduces an additional parameter G to represent the effective bandwidth for long messages, which is generally larger than that for short messages. The communication time for short messages is unchanged but is equal to $o + (k-1)G + L + o$ for long ones. This simple distinction between short and long messages was extended in [17] 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, the transport layer, and the network layer. 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. While such models may account for most of the idiosyncrasies of a parallel machine, they become quite difficult to use to design algorithms. For instance, they assume that senders and receivers synchronize and include that synchronization cost in the overhead. However, some MPI implementations use schemes that may not require synchronization, depending on message size.

Finally, Ino *et al.* proposed in [16] 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 `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)). Regarding message transmission time, the s parameter is used to distinguish small messages from large messages and is thus continuously piece-wise linear in message size (Figure 1(c)).

To summarize, the main characteristics of these models derived from the seminal LogP 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 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.

An alternative to expensive 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 interactions with every other flow. It generally amounts to solving a bandwidth sharing problem, i.e., determining how much bandwidth is allocated to each flow. While such models are rather flexible and account for many non-trivial phenomena (e.g., RTT-unfairness of TCP or cross-traffic interferences) [28], they still ignore protocol oscillations, slow start, and more generally all transient phases between two steady-state operation points. Some specific examples of unrepresented phenomena that can have major impacts on performance include saturation of the network to the point congestion would cause packet loss, and massive reemissions conditioned by timeout events.

These models have not been used extensively in the HPC field so far for at least two reasons. First, until a few years ago contention could be neglected for high-end machines. Second, such models are quite complex to implement and are often considered as too slow and complex to scale to large platforms. As we demonstrate, neither of these assumptions remains true today, and flow-based approaches can lead to significant improvements over classical LogP models.

While the aforementioned approaches model point-to-point communications, the modeling of collective communications is another critical point when studying MPI applications. Some projects model them with simple analytic formulas [2, 27]. 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 [13]. 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. Finally, it does not allow for performance extrapolation on a larger machine with similar characteristics. A third approach consists in tracing the execution of collective operations and then replaying the obtained trace using one of the aforementioned delay models [15, 32]. However, current implementations of the MPI standard dynamically select from up to a dozen different communication algorithms when executing a collective operation. 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 [32], LAPSE [8], MPI-SIM [3], or the work in [25] rely on simple delay models (affine point-to-point communication delay based on link latency and bandwidth). Other tools, such as Dimemas [2], LogGOPSim [15], or PHANTOM [31] 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 [22] that relies on OMNeT++. Finally PSINS [27] and PEVPM [13] provide complex custom models derived from intensive benchmarking to model network contention.

3. 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 envi-

ronment 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 tool kit. 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, carries with it high overheads, and for many cases it can be a more resource intensive approach 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 first modified to add monitoring and instrumentation code if necessary, and is run 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**. For distributed applications that run in parallel across multiple computational resources, it may be necessary to combine individual traces into a single unified trace file, in which case it is very important to make sure that any clock skew is properly accounted for so that the traces are correctly synchronized and the true order of events is preserved. 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 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 [15, 27, 21, 31, 14] 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 [9, 15], 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. 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 partial 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 provide this facility [22, 8, 3, 25].

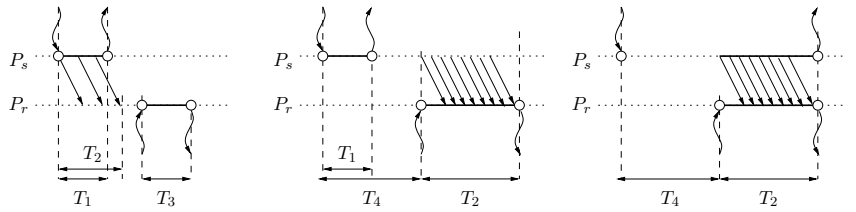
In a previous work [4] we showed that this partial online approach (coupled to a precise model of the platform) is only slightly slower than LogGoPSim, a performance-oriented offline simulator that disregards contention effects and only models the network latency. To support simulations at very large scale, SMPI provides a method of trading off accuracy for performance by benchmarking the execution of the program on some nodes, while skipping it on others 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 not containing large numbers of branching statements within the parallel code) 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, particularly in engineering or the sciences. These features are disabled by default, and have to be enabled by an expert user by adding macros and 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.

4. A "HYBRID" NETWORK MODEL

In this section, we report the 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. 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. A full description of the interconnection network is available online [12].

4.1 Point-to-point communications' model

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 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 2, this model is unable to account for the full complexity of a real MPI imple-



(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.

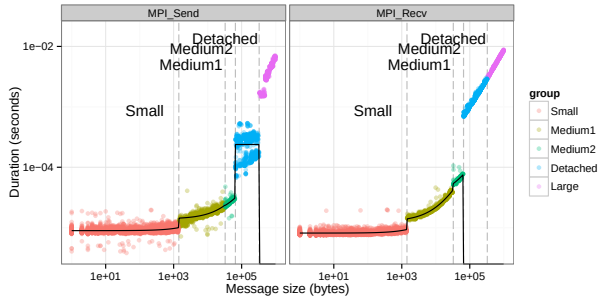


Figure 2: `MPI_Send` and `MPI_Recv` duration as a function of message size.

mentation. The measurements presented in Figure 2 were obtained according to the following protocol: To avoid any measurement bias, 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 of 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 $1,420 < k \leq 10,000$ or $10,000 < k \leq 65,536 = S_a$): these messages are still sent asynchronously but incur a slight overhead compared to small messages, hence a discontinuity at $k = 1420$. The distinction at $k = 10,000$ 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 is harmless and 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 2, 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 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 5.

4.2 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, 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, 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

[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

or cluster-like environment the mutual interactions between send and receive operations cannot safely be ignored.

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 pings from 1 to 19, i.e., half the size of the cabinet. As the network switch is well dimensioned, this experiment fails to 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 pings. We increase the number of concurrent 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 models, would suppose that $B_s + B_r = B$ on average since both directions strictly alternate. A multi-port model, as assumed by delay models, would estimate that $B_s + B_r = 2 \times B$ since communications do 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 processor 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 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. 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 [12], 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 is captured by describing the interconnection of two cabinets as three distinct links (uplink, downlink, and *limiting* link). The bandwidth of this third link is set to 13 Gb as measured.

This topology model could certainly be further refined. One intuitive lead is to consider the bandwidth limitation due to the switch backplane that may be reached during a full cluster bipartite communication. But such a refinement seems to be pointless as the experiments presented in Section 5 tend to show that this limit is never reached, or is at least hidden by other inaccuracy sources, such as packet dropping.

4.3 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 [10]. 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.

Simulation of the most commonly used algorithms for collective communications is not very difficult as most are rather simple (in concept if not necessarily in detail). However, capturing the selection logic of the various MPI implementations is much more complex. This logic is highly dependent on the implementation and generally embedded deep within the source code.

SMPI implements most of the collective algorithms by reusing the relevant source code of Star MPI [10]. It also provides a simple algorithm selector that mimics the behavior of OpenMPI in most cases. It was devised using both code inspection and execution trace comparisons. While sufficient in most cases, it leaves room for further improvements. In future work, we hope to provide additional selection algorithms, ideally one for each available MPI implementation. Then, we will provide a generic way to specify the selection algorithm to use, enabling the tuning of collective algorithms within SMPI. Finally, we would like to provide a semi-automatic benchmarking solution to discover the selection algorithm used in a given MPI implementation by extensive testing.

5. MODEL [IN]VALIDATION STUDY

5.1 Setup Description

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 [26]. 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 4, the *LogGPS* model that supersedes all the delay-based models, and a *fluid* model that is a basic linear flow-level model.

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.

5.2 Collective Communications

We begin our validation process with the study of an isolated MPI collective operation. We focus on the `MPI_Alltoall` function call that, as its name says, sends data from all to all processors. We selected this 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 collective operations. Toward this end, we enforce the use a single algorithm, i.e., the ring algorithm, for all message sizes. We ran the experiment five times and only kept the best execution time for each message size. Indeed, we noticed that first communication is always significantly slower than those that follow, which tends to indicate that TCP require some warm-up time. This phenomenon has also been noticed in experiments assessing the validity of the BigSIM simulation toolkit. The same workaround was applied.

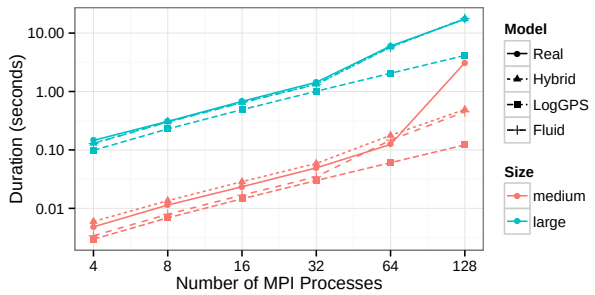


Figure 4: Comparison between simulated and actual execution times for the `MPI_Alltoall` operation.

Figure 4 shows the evolution of simulated and actual execution times for the `MPI_Alltoall` operation when the number of MPI processes varies for the three contention 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. In opposition to *LogGPS*, these two models account for network topology and contention, even when transferring data across cabinets (for 64 and 128 processes). Unsurprisingly, the *LogGPS* model is overly optimistic in such setting and completely underestimates the overall execution time. 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 prediction error under 5% 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 be explained by packet dropping in the main switch that leads to timeouts and re-emissions, hence incurring significant delays. For larger messages, TCP seems to be more stable, and if packet are dropped it has little influence on the larger total execution time.

5.3 NAS Parallel Benchmarks

The NAS Parallel Benchmarks (NPB) are a suite of programs commonly used to assess the performance of parallel platforms. From this benchmark suite, we selected three applications with different characteristics. The DT (Data Traffic) benchmark is a communication intensive application. In its Black Hole variant, DT consists in sending large messages of several MiB of data from multiple sources towards a single sink. DT implies many long lasting communications, but its scheme tends to prevent contention. For DT the class refers not only to the size of the problem but also to the number of communicating processes: classes A, B, and C involve 21, 43, and 85 processes respectively. 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. This pattern is very sensitive to noise as each process has to wait for the reception of a message before sending its own data. The third 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 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.

Figure 5 shows the comparison of the actual execution times measured on the *graphene* cluster for various instances of the three NPB benchmarks with the simulated times obtained with the studied models.

For the DT benchmark, which is composed of a few well distributed communications of large messages, the three models lead to comparable results and give good estimations of the actual execution times. These results confirm the observations made for the All-to-All collective operations. To some extent the DT benchmark corresponds to a reduction. The *LogGPS* model always underestimates the execution time, while the accuracy of the *fluid* model increases along with the size of the problem. Our *hybrid* is the most accurate across the whole set of experiments.

For the LU benchmark we can see that for the class C instances the three models lead to similar predictions and correctly estimate the evolution of the execution time. A more interesting discrimination occurs for the smaller class B instances. As mentioned earlier, the simulation of small messages is more problematic for the *fluid* and *LogGPS* models. Their representation of asynchronous sends is too simple to be accurate, especially for the *fluid* model. This iterative benchmark alternates computations and communications at a high rate. Similar or close completion times as shown by Figure 5 may hide a bad, but lucky, estimation of each component of the execution time. To be sure that estimation errors on computation times are not compensated by errors on the communication times, we propose to compare (a part of) the Gantt charts of the actual and simulated executions. Such a comparison is given by Figure 6 that shows a period

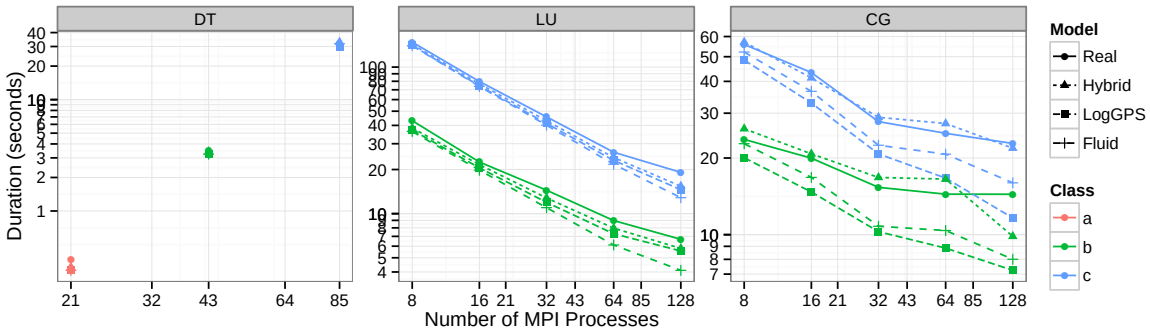


Figure 5: Comparison between simulated and actual execution times for three NAS parallel benchmarks.

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.

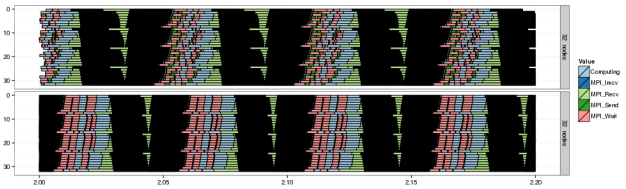


Figure 6: Part of the Gantt chart of the execution of the LU benchmark with 32 processes. The upper part displays the actual execution while the lower is the simulation with the *hybrid* model.

Such Gantt charts are difficult to align perfectly due to the size of the initial execution traces and clock skew that requires a synchronization for the real trace. Real executions are also subject to system noise that does not occur in simulation causing some small discrepancies related to the computation time. This then prevents the observation of a perfect match. However, Figure 6 clearly shows that despite the small time scale, the simulation correctly renders the general communication pattern of this benchmark.

Finally, the results obtained for the CG benchmark are even more expressive. Indeed, this benchmark transfers messages that never fall in the *large* category. It is then completely unfavorable to the *LogGPS* and *fluid* models. Conversely, our *hybrid* model that distinguishes four different classes of "small" messages produces good estimations for up to 64 processes. There is a large difference between the actual execution time and the simulated time obtained with the *hybrid* model when executing a class B instance with 128 processors, and to a lesser extent for the larger class C. Then we have to determine if the source of this gap comes from a bad estimation by the model or a problem during the actual execution. Our main suspect was the actual execution, which is confirmed by the Gantt chart presented in Figure 7.

The execution time is 14.4 seconds while the prediction of the *hybrid* model is only of 9.9 seconds. We see two outstanding zones of `MPI_Send` and `MPI_Wait`. Such operations

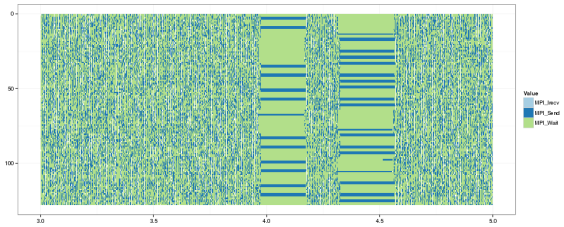


Figure 7: Two seconds Gantt-chart of the real execution of a class B instance of CG for 128 process.

typically take few microseconds 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 prediction of the *hybrid* model. The same phenomenon was for class C. This behavior of the interconnection network can be considered as a bug that needs to be fixed in a production environment.

5.4 Sweep3D

Here we consider another benchmark representative of the heart of real applications. The Sweep3D kernel solves a time-independent discrete ordinates 3D Cartesian geometry neutron transport problem. This Fortran code makes heavy use of pipelining techniques, and thus overlaps communications with computation. Eight waves traverse the 3D grid, each starting from one corner and moving towards the opposite corner. In terms of MPI communications it corresponds to a complex pattern of point-to-point communications, with some collective operations. In our experiments, the Sweep3D application is simulated on-line by SMPI.

Figure 8 presents the achieved results for two small instances of the Sweep3D benchmark. With regard to the previous experiments, these results show two interesting differences. First, the measured execution times are very short, less than 1.4 seconds for the longest run. It allows us to see whether models can be distinguished even over such a short time pe-

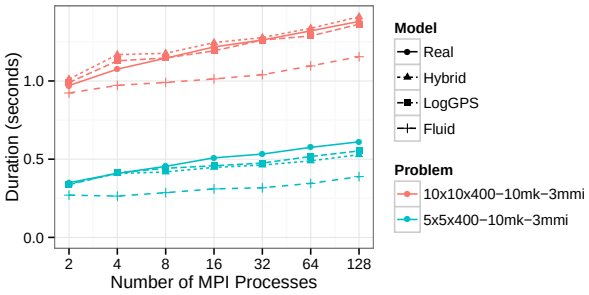


Figure 8: Comparison between simulated and actual execution times for two Sweep3D instances.

riod. Second, such small instances of the benchmark do not scale at all and are even badly impacted by the addition of extra processors. This then tells us if simulation can be useful not only to predict good and expected scalability results, but also to detect when a scalability limit has been reached or exceeded. The answer to both questions is yes. We see that small messages are badly handled by the simple *fluid* model, which is far less accurate than the *hybrid* and *LogGPS* models. These two models provide very similar estimations of the execution time. It is also interesting to note that all three models, including the imprecise *fluid* models, are able to reflect the increasing trend of the execution time. This shows the benefits of partial on-line simulation to analyze the performance of MPI applications.

6. SIMULATING A REAL APPLICATION

In this section we aim at demonstrating the capacity of SMPI to simulate a real, large, and complex MPI application. BigDFT is a Density Functional Theory (DFT) code that develops a novel approach for electronic structure simulation based on the Daubechies wavelets formalism [11]. The code is HPC-oriented, i.e., it uses MPI, OpenMP and GPU technologies. So far, BigDFT is the sole electronic structure code based on systematic basis sets which can use hybrid supercomputers. 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 is determined by the problem size.

While this application can be simulated by SMPI without any modification to the source code, its large memory footprint means that to run the simulation on a single machine would require an improbably large amount of system ram. Applying the memory folding techniques mentioned in Section 3 and detailed in [6], 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 with 12GB of memory. Figure 9 shows the execution trace corresponding to this simulated execution.

In addition to this particular application, we were also able to simulate another geodynamics application called SpecFEM3D [23] and the full LinPACK suite. SMPI also is tested upon 80% of the MPICH test suite every night to make non-regression tests. We can thus claim that SMPI is

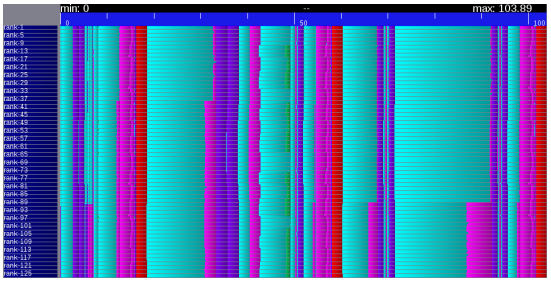


Figure 9: Simulated execution trace of BigDFT with 128 processes.

not limited to toy applications but can effectively be used for the analysis of real scientific applications.

7. CONCLUSION AND FUTURE WORK

We have demonstrated that accurate modeling and performance prediction for a wide range of parallel applications requires proper consideration of many aspects of 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 have described the implementation of a proposed 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 using competing simulation toolkits.

Our priority in this work was the validation of the model at a small scale and for TCP over Ethernet networks. Such a tool would prove very useful for efforts such as the European Mont-Blanc project [19, 24], which aims to prototype 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.

While experimental data have been collected by hand for the present analysis, the need to collect more and more traces to test against various platforms is a strong incentive to automate this task. Thus, one future work will be to build a systematic validation/invalidation framework based on the automatic collection of traces.

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 [30]. Accordingly, SMPI and all the software stack are provided as open-source software available for download from the public Internet. We also intend to distribute the MPI benchmarks used to measure platform parameters and check model validity, and the R scripts exploiting the resulting data along with the next packaged version of SimGrid.

8. 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).

9. 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] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] C. Ernemann, B. Song, and R. Yahyapour. Scaling of workload traces. In *Proc. of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 166–182, June 2003.

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

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] The mont-blanc project. http://www.montblanc-project.eu.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

ScalA '11. ACM, 2011.

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

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

[27] 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.

[28] 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. Under revision in TOMACS.

[29] J. Xu, M. Maxey, and G. Karniadakis. Numerical simulation of turbulent drag reduction using micro-bubbles. *Journal of Fluid Mechanics*, 468:271–281, 9 2002.

[30] 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.

[31] 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.

[32] 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.