Parallel Processing Letters

MODELING THE PERFORMANCE OF COMMUNICATION SCHEMES ON NETWORK TOPOLOGIES

JAN LEMEIRE

and

ERIK DIRKX

and

WALTER COLITTI

ETRO Department, Vrije Universiteit Brussel, Pleinlaan 2
B-1050 Brussels, Belgium

Received October 2007
Revised March 2008
Communicated by C. R. Jesshope

ABSTRACT

This paper investigates the influence of the interconnection network topology of a parallel system on the delivery time of an ensemble of messages, called the communication scheme. More specifically, we focus on the impact on the performance of structure in network topology and communication scheme. We introduce causal structure learning algorithms for the modeling of the communication time. The experimental data, from which the models are learned automatically, is retrieved from simulations. The qualitative models provide insight about which and how variables influence the communication performance. Next, a generic property is defined which characterizes the performance of individual communication schemes and network topologies. The property allows the accurate quantitative prediction of the runtime of random communication on random topologies. However, when either communication scheme or network topology exhibit regularities the prediction can become very inaccurate. The causal models can also differ qualitatively and quantitatively. Each combination of communication scheme regularity type, e.g. a one-to-all broadcast, and network topology regularity type, e.g. torus, possibly results in a different model which is based on different characteristics.

Keywords: Parallel Processing, Performance Modeling, Causal Inference, Network Topology, Structure.

1. Introduction

The goal of this work is to study the impact of structure in *network topology* (NT) and communication on the time the system needs to exchange data among processors. We call the ensemble of messages sent among processors the *communication scheme* (CS) of a parallel program. We will experimentally demonstrate that regularities in NT and CS greatly affect the performance, qualitatively and quantitatively. Consider a star NT, where every processor is connected to one central processor, or a ring NT, in which the connections of

the processors form a ring (Fig. 1). Consider a structured CS such as a one-to-all broadcast or a shift collective communication. In the former one process sends a message to all other processes, where in the latter every process sends a message to its direct neighbor (Fig. 1). It is intuitively clear that a one-to-all broadcast communication on a star network and the shift operation on a ring network are very efficient. In contrast, a broadcast on a ring or a shift on a star results in a much higher runtime. The efficiency by which communication can be completed depends on the *match* of CS and NT.

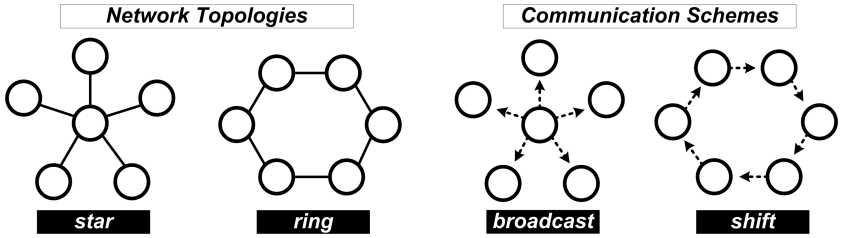

Fig. 1. Two regular network topologies and two regular communication schemes.

The patterns in CS and NT affect which and how variables affect the communication time. We propose causal models [13] to explicitly describe the relations among the variables of interest. Causal structure learning algorithms [16] are employed to infer models for the causal dependency of communication time on properties of CS and NT. The analysis is based on experimental data retrieved from simulation of the communication on the network.

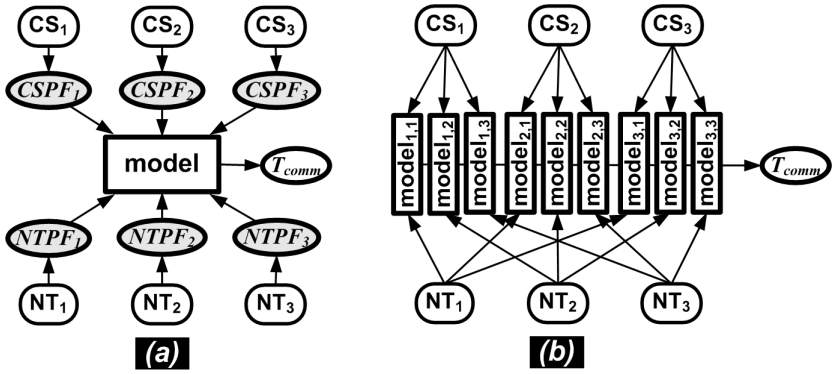

Fig. 2. Network Topologies and Communication Schemes can be characterized by generic performance properties (a) or every combination entails a specific model (b).

Next, we investigated the existence of a generic performance prediction model. The question is under which circumstances and to what extent can communication schemes and network topologies be characterized by generic performance properties. Such that the communication time, T_{comm}, can be predicted by a simple model based on these characteristics, as shown in Fig. 2(a). Each NT_i is characterized by a Performance Factor, called

$NTPF_i$, and each CS_j is characterized by Performance Factor $CSPF_j$. Ideally, a function of the form $T_{comm} = f(NTPF_i, CSPF_j)$ exists that gives accurate estimations for the communication time for any combination of CS_j and NT_i. We showed experimentally that random NTs and random CSs can be characterized by a single performance factor.

This model can, however, become very inaccurate when patterns occur in CS or NT. Non-random NTs or CSs, exhibiting certain patterns, can differ not only quantitatively, but also qualitatively by the properties of NT and CS influencing the performance. In the worst case, each combination of NT regularity type and CS regularity type would need a different model, as shown in Fig. 2(b). Our results show that it is not possible to characterize the performance of an NT while not considering the patterns of the CS. An NT can perform well for one CS, but not for another.

This paper is organized as follows. The next chapter discusses related work, section 3 the causal learning algorithms and section 4 the experimental setup. Section 5 gives the causal models learned for random communication on random networks, and compares it with models for structured communication on regular networks. Finally, in section 6 we define a performance factor that characterizes individual NTs and CSs, and experimentally test how well communication time can be predicted based on these factors.

2. Related Work

Our experiments adopt the setting used in [9, p. 106]. Kumar et al. built analytical formulas for the performance of structured collective communication operations, such as the one-to-all broadcast, all-to-all or shift, on regular interconnection networks, such as the hypercube, star or mesh. We automatically derive models from experimental data.

Our study of the genericity of performance models is similar to that based on the *convolution method* [14]. Its goal is the prediction of the runtime of an application on an arbitrary system. This requires the detection and definition of independent application and system performance characteristics - called *application signature* and *system profile* - and a simple functional relation to calculate from both the performance of an application running on a system. The separation of signatures and profiles means that signatures need only be gathered once to support prediction on multiple machines. Fig. 2 depicts the two extremes: does one model and generic characteristics suffice for accurate performance estimation or does every combination require a different model? A similar strategy is employed by Marin and Mellor-Crummey [12]. They also try to separate the contribution of application-specific factors from the contribution of architectural characteristics to overall application performance. Results indicate, however, that the use of a single, simple synthetic metric, or a linear combination of such simple metrics, to predict the performance of high-performance application performs poorly [3]. Our research demonstrates that performance characteristics depend on the patterns in application and system. Simple models can only be realized when limited to specific regularities.

Our approach falls under the category of tools that monitor performance by collecting statistical data of multiple experiments. It is concerned with counts and durations. Current tools that support multiple experiment analysis plot performance variables (PMaC [15],

PERFORM [8] and SCALEA [19]) and inefficiencies (Aksum [5]) as a function of application and system parameters. The other approach is to study event traces, according to which the exact sequence of actions that took place during a run is recorded. Statistical data is more compact than event traces, but it is already noticed that the predictive power is limited [3]. Better predictions can be acquired by simulating the recorded trace on a model. This approach is limited since it doesn't give parameterized models. It also provides little insight into the factors that determine the overall performance.

Also in the study of the performance of the internet it is discovered that the development of accurate models poses many problems. On the one hand is the interconnection topology not random or a simple planar graph, but a constantly changing heterogeneous combination of regular structures [6][21][2]. On the other hand, internet traffic at the level of IP packages resembles more self-similar than Poisson processes [6]. Our results show that regularities in network or traffic can give different quantitative and qualitative results. Modeling these regularities is therefore required to obtain accurate simulations.

3. Causal Structure Learning

This section will briefly introduce causal models and the main learning algorithm.

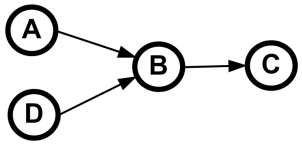

Fig. 3. Example causal model.

3.1. *Graphical Causal Models*

Causal models [13] [16] [18] intend to describe with a Directed Acyclic Graph (DAG) the structure of the underlying physical mechanisms governing a system under study. The state of each variable, represented by a node in the graph, is generated by a stochastic process that is determined by the values of its parent variables in the graph. They are the *direct causes*.

Fig. 3 depicts a simple causal model in which A and D are the direct causes of B and B is the direct cause of C. A and D are indirect causes of C. The theory is based on the observations that a causal structure implies *conditional independencies*. This is best understood by the *Markov property*. From the causal structure of Fig. 3 it follows that A is independent from C by conditioning on B. A conditional independency is denoted by the ternary operator $.\perp\!\!\!\perp. \mid .$ and is defined as

$$A \perp\!\!\!\perp C \mid B \Leftrightarrow P(C \mid A, B) = P(C \mid B). \tag{1}$$

The conditional independence expresses that learning the value of A does not provide additional information about C once the state of B is known. The same independency follows

from causal structures $C \rightarrow B \rightarrow A$ and $A \leftarrow B \rightarrow C$. A v-structure, on the other hand, such as $A \rightarrow B \leftarrow D$ in the model of Fig. 3 (the middle node, B, has two incoming edges), is characterized by an unconditional independency of A and D, $A \perp\!\!\!\perp D$, and a conditional dependency, $A \not\perp\!\!\!\perp D \mid B$. A and D are initially unrelated but become dependent when conditioned on their mutual effect, B. Variable B is called a *collider* on the path between A and D.

A graphical criterion, called d-separation, was defined which allows the retrieval of all conditional independence that follow from a causal structure [7]. The principle of the criterion is that two variables are dependent if they are connected in the graph with a path in which no variables are members of the conditioning set and which does not contain v-structures, unless the collider belongs to the conditioning set.

The conditional independencies that follow from a causal structure give the information to infer causal models from data. These independencies are irrespective of the precise parameterization of the causal mechanisms of the system.

3.2. *Causal Inference*

The *PC algorithm* is the basic constraint-based learning algorithm [16]. It is part of the TETRAD tool [17], developed by the Dept. of Philosophy of Carnegie Mellon University and freely available. The algorithm consists of two steps. First it constructs an undirected graph by finding the direct relations. Thereafter, the algorithm tries to direct the edges using orientation rules. The flexibility of the algorithm allows for the insertion of background knowledge. The user can specify edges that are required or forbidden. He can also put constraints on orientations. In the context of performance models, application and system parameters are designated as input nodes that can only have outgoing edges.

The first step, called *adjacency search*, is based on the property that direct relations cannot become probabilistically independent upon conditioning on some other set of nodes. Adjacent variables share exclusive information, while indirectly related variables become independent by conditioning on some other variables which lie on the causal paths between both variables. The different steps in the learning of the model of Fig. 3 are illustrated in Fig. 4. The algorithm starts with a fully-connected undirected graph (3a). First it removes all edges between uncorrelated variables (3b). Then all edges for which a conditioning set can be found that renders both variables independent (3c). All subsets of variables are checked for conditional independencies. If a test is successful, the edge is removed. The algorithm starts by checking unconditional dependencies and then gradually adding nodes to the conditioning set up to a certain maximal number (set to 2 in our experiments). It selects the nodes in such a way as to minimize the number of tests it has to perform. This result in an undirected graph (3d).

In the second step the PC algorithm tries to direct the edges using orientation rules. These rules are based on the detection of *v-structures*. If three variables are connected by two edges, for example $U - V - W$, there are four possibilities to orient both edges. The v-structure, $U \rightarrow V \leftarrow W$, is unique among these, since U and W are initially independent, but become dependent by conditioning on V. The opposite is true for the three other

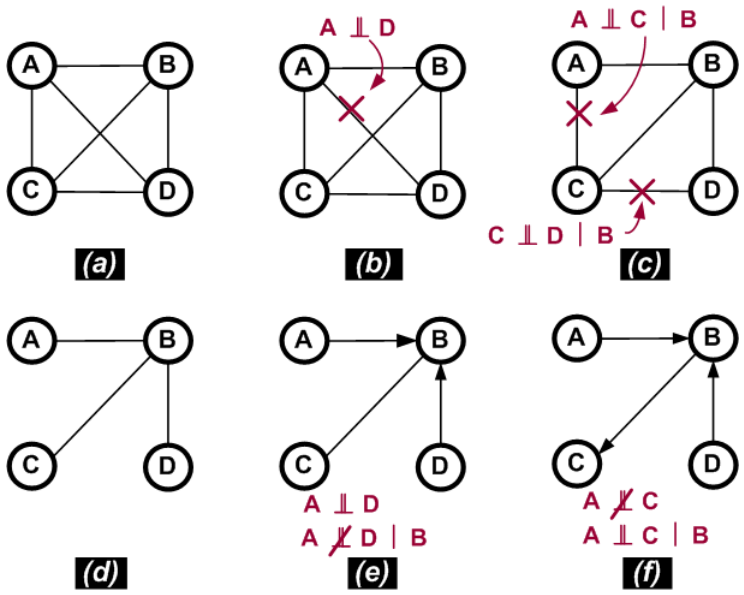

Fig. 4. Inference of the example causal model of Fig. 3 by the conditional independencies it entails.

orientations. Applied on the undirected graph of Fig. 4, v-structure $A \rightarrow B \leftarrow D$ is detected (3e). Finally, the $B - C$ relation can be oriented as $B \rightarrow C$ (3f), since an opposite orientation leads to v-structure $A \rightarrow B \leftarrow C$ that cannot be confirmed by the independencies found in the data. In absence of enough v-structures, it might be that there is not enough information to direct all edges. The learning algorithm leads to a set of observationally indistinguishable models, which have the same undirected graph and v-structures. Fortunately, the orientation of the relations in performance models is straightforward in most cases. Knowledge of the input and output variables allows orientation of the edges connected to it. Any further doubt about the orientation of some edges should be resolved by a human expert.

3.3. *Independence Test*

The learning algorithms are based on the information provided by the conditional independencies found in the data. TETRAD uses Pearson's correlation coefficient for calculating the dependency of continuous variables. It gives a measure of how close a relationship approximates linearity. Correlations can measure non-linear relations, as long as they are quasi-monotonically increasing or decreasing. Partial correlations, however, fail if the relations diverge too much from linearity. As correlations in computer systems performance metrics are strongly non-linear, the utility of Pearson's coefficient is of limited use. The solution lies in the information-theoretic concept of *mutual information* [4]. It quantifies, independently of the form of the relation, the degree of association between variables. We also extended the TETRAD toolset such that data consisting of a mixture of continuous

and discrete variables can be treated, which is currently not possible. The estimation of the underlying probability distribution for continuous variables is based on a *kernel density estimation* [20]. After having quantified the degree of association, a threshold is employed in order to decide upon dependence or independence. Details about the calculation of the mutual information and the calibration of the kernel density estimator are given in [11].

3.4. *Deterministic Relations*

Fig. 5. Example model with a deterministic relation, $Y = f(X)$.

Deterministic relations pose a problem for constraint-based algorithms. Take the model of Fig. 5. The relation between X and Y is a function: $Y = f(X)$. Deterministic variables are usually depicted with double-bordered circles. The model implies, by the Markov property, that

$$X \perp\!\!\!\perp Z \mid Y \tag{2}$$

but from the function it also follows that

$$Y \perp\!\!\!\perp Z \mid X \tag{3}$$

for, by the functional relation, X contains all information about Y.

We call variables X and Y *information equivalent* with respect to reference variable Z if

$$X \not\perp\!\!\!\perp Z \;\&\; X \perp\!\!\!\perp Z \mid Y \;\&\; Y \perp\!\!\!\perp Z \mid X \tag{4}$$

Either variable becomes conditionally independent from Z by conditioning on the other.

Models from systems containing information equivalences will not be learned correctly by the PC algorithm. The first condition of Eq. 4 implies that X and Z should be related. The second condition states that X is only indirectly related to Z via Y. Yet the third condition implies the opposite, that Y should be related to Z via X. Consequently, the adjacency search step of the PC algorithm will fail in constructing such a model; it would remove both edges $X - Z$ and $Y - Z$. The solution we opt for is to connect among information equivalent variables the one with has the simplest relation with the reference variable [10]. From the perspective of information, X and Y are equivalent with respect to Z. Connecting both to Z would represent redundant information and in this manner disrupt the minimality condition. If complexities match, as in the case of a linear relation between X and Y, the one that is the cause of the other is taken. If this does not lead to a definite choice, it is left over to the human expert.

The complexity of relations is measured with a regression analysis, which makes a trade-off between model complexity and error. Details of our extension to the PC algorithm are explained in [10].

4. Experimental Setup

We will experimentally investigate the execution time of an ensemble of messages, called communication scheme (CS), on network topologies (NT). The experimental data is retrieved from simulations. Random schemes and random topologies will be considered, but also schemes and topologies having a certain structure. To concentrate on the effect of structure on the performance we will make some simplifying assumptions. Our research is limited to homogeneous network topologies with bidirectional links all having identical latency and bandwidth. The time a message takes to make one hop is approximated by $latency + messagesize/bandwidth$. The simulation model of the network is described by a graph in which each node represents a processor and each edge a communication link. A communication scheme is a set of messages with a certain source, destination and message size. The message size is chosen to be the same for all messages, since its influence on the execution is not a part of our investigation. The message hop time is thus a constant. It is set to 1 time unit. Messages take the shortest path to go from source to destination. Messages are queued if the link through which they have to pass is occupied. The simulator is a discrete event simulator [1], the operation of a system is represented as a chronological sequence of events. Each event occurs at an instant in time and marks a change of state. In simulating network traffic, each hop a message makes is an event.

The following graph types are considered for the network topologies, each of them having specific parameters:

- **Random graph**: is constructed by probabilistically adding edges to a given set of nodes. Parameters are the number of nodes, $nodes$, and the relative node connection degree, $relConnect$, defined by the average number of links of a node divided by the total number of nodes. It is ensured that, indirectly, all nodes are connected so that there exists a path between any two nodes.
- **Ring**: is a graph in which all nodes are connected so that they form a ring. Its parameter is the number of nodes, $nodes$.
- **Torus**: is a closed surface defined as the product of two rings. Parameters are the number of nodes of the first ring, $nodesR_1$, and that of the second ring, $nodesR_2$.
- **Star**: is a graph that consists of one node in the middle which is connected to all other nodes. Its parameter is the number of nodes, $nodes$.
- **Neighbor graph**: is a graph in which nodes are randomly positioned in a plane and randomly connected but with a higher probability for neighbor nodes. The probability of having an edge between two nodes decreases quadratically with the distance between them. The parameter is the relative node connection degree, $relConnect$. It is the same to that of a random graph.
- **Planar graph**: is a graph in which nodes are randomly positioned in a plane and randomly connected, but in such way that none of the edges intersect. The sole parameter is the relative node connection degree, $relConnect$.

Each type is represented by a set of instantiations, generated by randomly chosen parameter values. The values are picked out of the following ranges according to a uni-

form distribution: $nodes \in [20, 100]$, $relConnect \in [0.05, 0.9]$ and, for the torus, $nodesR_1 \in [4, 14], nodesR_2 \in [4, 14]$.

Considered communication schemes and their parameters are:

- **Random communication**: each node sends a number of messages, randomly chosen with average $nodeMsgs$, to randomly selected destinations. The range of the parameter is: $nodeMsgs \in [20, 100]$.
- **One-to-all broadcast**: one node, the node with index 0, sends a personalized message to each of the other nodes.
- **All-to-all broadcast**: each node sends a personalized message to each of the other nodes.
- **Shift**: each node i sends a message to the node with index $i + indexShift$. Parameter $indexShift$ is chosen from range $[1, 5]$.

Variables that are measured during each experiment are the following:

- **Graph properties**: average distance, $avgDistance$, between two nodes (distance is the minimal number of hops required to get from one node to another) and $diameter$, which is the maximal distance between any two nodes.
- **Communication properties**: average number of messages per node, $avgNodeMsgs$.
- **Overall performance**: total time to complete all communication, $commT$, the runtime divided by the average number of messages per node, $timePerNodeMsgs$, average time it takes for a message to arrive at its destination, $avgTravelT$, average number of hops of a message, $avgHops$, maximal number of hops of a message, $maxHops$, and average time a message is queuing, $avgQueueT$.

5. Qualitative Performance Models

In this section we apply the PC learning algorithm on the experimental data from simulation of the execution of a communication scheme on a network topology. First a causal model is inferred from the data of a set of 200 couples of random CSs and random NTs. Next, the models for all combinations of CS types and NT types are explored.

5.1. *Random Communication Schemes on Random Network Topologies*

Experimental data of random communication on random interconnection networks results in the model shown in Fig. 6. The communication time is linearly dependent on the number of messages per node, $avgNodeMsgs$, hence it made sense to introduce the ratio $timePerNodeMsgs$. It gives a quantity that is independent of the number of messages and therefore reduces the complexity of the model. Deterministic variables, which are a function of its parents in the graph, are depicted with double-bordered circles. The average number of hops, $avgHops$, is determined by the average distance, $avgDistance$, of the graph. The model also reveals that the variable $avgDistance$ gives maximal information

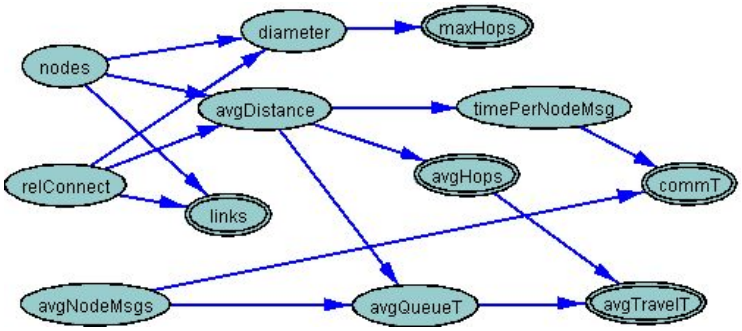

Fig. 6. Model of communication performance of random communication on a random topology.

about the time per node message. $avgDistance$ is, however, almost completely determined by $relConnect$, except for low values of $relConnect$. With a low number of links, the graph's randomness makes that the average distance can vary a lot from graph to graph. Variable $relConnect$ thus also greatly influences $timePerNodeMsgs$, but does not capture all information about it. Variable $avgDistance$ does. This illustrates the Markov property (Section 3.1): $relConnect \perp\!\!\!\perp timePerNodeMsgs \mid avgDistance$.

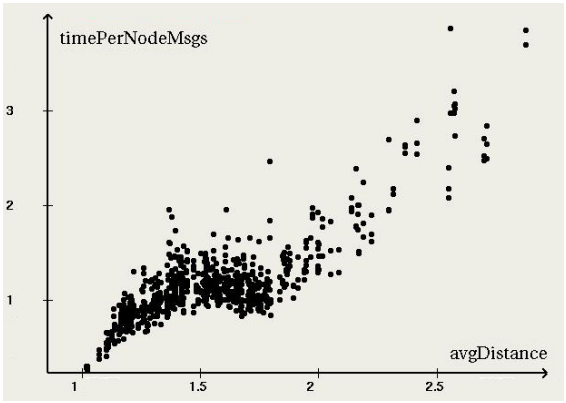

Fig. 7. Communication time per number of messages per node versus average distance for random Network Topologies and random Communication Schemes.

Fig. 7 shows the statistic $timePerNodeMsgs$ as a function of $avgDistance$. For any given parameter configuration, $nodes$, $relConnect$ and $avgNodeMsgs$, there is still a high uncertainty on the communication time. In the next section, a quantitative model is proposed to characterize individual NTs and CSs so that the uncertainty can be reduced.

5.2. *Regular Communication Schemes on regular Network Topologies*

The models that are learned for the combinations of CS and NT classes give models that are sometimes quite different than that of complete randomness (Fig. 6). The following table

summarizes these qualitative differences. Symbol '=' denotes that the model corresponds with the model for random communication on random networks, symbol '#' that they are different. When different, the numbers in brackets explain the differences.

	random graph	*torus*	*neighbor*	*planar*	*star*	*ring*
random CS	reference model	# (6)	=	= (1)	=	=
broadcast	# (1)(3)(4)	# (4)	# (5)	= (1)	# (8)	= (2)
all2all	=	# (6)	=	=	# (9)	=
shift	= (1)(3)	# (7)	= (5)	=	# (8)	# (7)

Explanation of the differences:

- (1) The relations have the same shape, but there is a higher uncertainty on the relations.
- (2) All relations are deterministic.
- (3) The relation $avgDistance$ - $avgHops$ is not deterministic.
- (4)The number of nodes, $nodes$, also influences $timePerNodesMsgs$.
- (5) Identical to model of random communication with the same CS.
- (6) Besides $avgDistance$, the communication time is less optimal the more $nodesR_1$ and $nodesR_2$ differ. If $nodesR_1$ is low, there exist only a few paths connecting the circles of the second dimension. This leads to congestion in these paths. A variable corresponding to the absolute difference between both, $|nodesR_1 - nodesR_2|$, was added to verify this.
- (7) The index shift, $indexShift$, also determines performance. Moreover, for a ring network it is the only parameter affecting the performance variables $timePerNodesMsgs$, $avgHops$ and $maxHops$.
- (8) All performance variables have a constant value.
- (9) $timePerNodesMsgs$ is a constant, $nodes$ affects $avgQueueT$.

These results show clearly that the model of Fig. 6 becomes invalid. But it is difficult to draw general conclusions. Random communication behaves quite similar for all kinds of graphs, except for a torus, in which case the average distance is not the sole direct cause of $timePerNodesMsgs$. The neighbor graph gives the same models as for the random graph. But the table shows that specific regularities in the communication can result in very specific matches, such as on star or ring graphs. The correctness of the learned models can be verified when one reasons about how the execution of a specific regular communication on a structured topology behaves. For instance, consider a shift collective communication performed on a ring which results in a very efficient execution. The transmission happens synchronously. Each transmission channel is occupied by exactly one message at each time instance. No message has to queue before arriving at its destination. $indexShift$ deter-

mines the number of hops each message has to make and thus also the communication time.

6. Quantitative Performance Models

The previous section was devoted to qualitative models, providing insight into the performance. In this section we are interested in the prediction of the communication time. We will build a generic model, in the sense of Fig. 2(a) and the convolution method (see related work section), that adequately estimates the runtime of random communication on random topologies. A generic performance property is defined that characterizes individual NTs and CSs, and a simple function that calculates the runtime of any combination of NT and CS. In the subsequent subsection we investigate whether this quantitative model is also valid for combinations of regular CSs and NTs.

6.1. *Random Communication Schemes on Random Network Topologies*

We propose a definition of a Performance Factor for characterizing individual NTs and CSs, called respectively $NTPF$ and $CSPF$. Its value will be experimentally measured on a benchmark. The benchmark consists of 50 random graphs having 25 nodes and varying $relConnect$ and 50 random communication schemes having 25 messages per node. The benchmark average runtime, denoted $refT$, is the average runtime of all 50×50 combinations of the benchmarks NT and CS. It serves as a reference for characterizing the performance of NTs and CSs. The performance factor of topology i, $NTPF_i$, is defined as

$$NTPF_i = benchmarkT(NT_i)/refT \tag{5}$$

with $benchmarkT(NT_i)$ the average runtime of running graph i on the 50 benchmark communication schemes. This factor lies around 1. If it is smaller, it means that the communication finishes faster than on the benchmark graphs. If higher, the communication needs more time to complete. Likewise is the performance factor of CS_j defined by the average runtime of the simulation of the CS on the benchmark topologies, denoted $benchmarkT(CS_j)$:

$$CSPF_j = benchmarkT(CS_j)/refT \tag{6}$$

The function to calculate the runtime of running CS_j on NT_i is defined as follows:

$$prediction(CS_j, NT_i) = refT \times CSPF_j \times NTPF_i \tag{7}$$

Fig. 8 present the results for the comparison of the estimated with the real runtime for a test set of 100 random NTs and 100 random CSs. Although there is a difference, a correlation coefficient of 0.98 indicates that a good estimation is achieved.

6.2. *Regular Communication Schemes on regular Network Topologies*

Random communication on random topologies results in statistical values. Deviations of the average for specific instantiations are accurately modeled by the CS and NT performance factors. This section will uncover that the method based on the benchmark NTs and

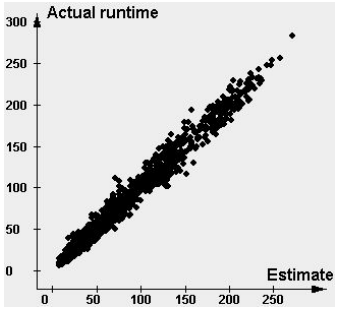

Fig. 8. Real versus estimated communication time for random communication on random topologies.

CSs is not always successful in predicting the performance of regular NTs and CSs. NTs containing 25 nodes are considered. The same equations, Eq. 5 and Eq. 6, were used for experimentally attributing a value to each CS and NT type. A set of instantiations for each type (see section 4) was benchmarked. The results are shown in the following table. The numbers come from averaging over the results of 50 different sets measured on 10 different benchmarks. It gives the average benchmark runtime, the average factor and the average standard deviation of the factor for the different instantiations in each set. The latter is an indication of how much the performance factor varies from one instantiation to another. Its value is not filled in if there is only one instantiation. The average runtime of the benchmark (random graphs), $refT$, is 46.9 time units. Recall that the benchmark consists of random graphs containing 25 nodes and 25 messages per node are communicated. 1 time unit is needed for a message to perform 1 hop.

	benchmarkT	*factor*	*stddev*
random graph	50	1.0	0.9
torus	123	2.4	1.3
neighbor graph	141	2.8	2.1
planar graph	191	3.8	1.7
star	77	1.5	-
ring	190	3.7	-
random	50	0.99	0.09
broadcast	8.7	0.17	-
all2all	44	0.87	-
shift	25	0.49	0.27

Most graphs have a factor that is more than 1, which indicates that the communication time is higher than that of random graphs. A low factor is not necessarily an indication of a bad efficiency when the number of edges is taken as the cost of the network. The structured graphs have fewer edges than the random graphs.

Similar results are obtained for graphs with a different number of nodes or a different number of messages. Except that for bigger graphs or more messages the standard deviation becomes smaller. Each experiment comes closer to the statistical average.

Once the factors have been established, the quality of the estimated runtime calculated with Eq. 7 can be verified for each combination of regularity type. We calculate the relative estimation error according to the ratio of the average sum of squared errors with the average runtime:

$$relative\ estimation\ error(NT_i, CS_j) = \frac{\sqrt{SSQ/n}}{averageT(NT_i, CS_j)} \tag{8}$$

with n the number of data points. The results are shown in the following table. The value in brackets gives the relative difference between the average estimated time and average real runtime:

	random graph	*torus*	*neighbor graph*
random	14% (+0.8%)	7% (-0.3%)	9% (-0.5%)
broadcast	**72% (+11%)**	**40% (-25%)**	**94% (-50%)**
all2all	12% (-0.2%)	6% (-6%)	10% (+6%)
shift	**61% (+5%)**	**270% (-160%)**	**97% (-46%)**
	planar graph	*star*	*ring*
random	8% (+0.2%)	8% (+1%)	6% (-0.4%)
broadcast	**95% (-70%)**	**1150% (-1150%)**	**39% (39%)**
all2all	9% (+6%)	**37% (-37%)**	5% (-5%)
shift	**90% (-65%)**	16% (-12%)	**330% (-250%)**

The standard deviation of the values over the 50 different experiments is about 7% of the given values. This shows that there is a quite large 95%-confidence interval for the values of about 15%. Nevertheless is there a big difference between combinations that give quite good estimations and those for which the prediction is completely wrong (put in boldface).

The standard deviation of the estimation of random CS on random NT is 14%. But the low value of 0.8% for the difference between average estimation and real time indicates that the estimation is unbiased. On the contrary, a broadcast on a planar graph executes consequently faster than expected. An average difference of -70% for a relative error of 95% shows that almost all estimations are too high. Several combinations lead to better matches than expected by the benchmark. While other combinations are well predicted, such as the *all2all* communication scheme. An *all2all* resembles random communication.

7. Conclusions

Besides the capacities of a communication channel, such as the latency and the bandwidth, it is also the topology of the interconnection network that affects the time a set of messages needs to be transmitted. Especially when Network Topology (NT) or Communication Scheme (CS) exhibit specific regularities. Causal inference algorithms were used to construct, from experimental data, causal models revealing the relations among the variables and the impact of every variable on the overall communication performance. The results show that no general conclusions can be drawn. The causal models can be radically different for specific combinations of CS regularities and NT regularities.

The same conclusion can be drawn for the prediction of the communication time. For attaining genericity, it must be possible to characterize an individual NT with a set of properties such that it allows accurate prediction of the communication time for any communication scheme that is executed on it. And vice versa, a set of generic properties must exist for characterizing any CS. We defined a property and a simple function to estimate the communication time for a combination of CS and NT. The property is measured on a benchmark which consists of a set of random CSs and random NTs. This method gives accurate results when the NT corresponds to a random graph and the CS is adequately modeled by messages with randomly chosen source and destination. Regular graphs also give good results when confronted with random communication, but not when combined with certain structured communication types.

Concluding, regularities in NTs and CSs cannot be ignored for effective understanding and prediction of communication performance. Good predictions for new topologies or communication schemes can only be made with models that apply for their structure.

References

[1] Jerry Banks, John Carson, Barry L. Nelson, and David Nicol. *Discrete-Event Simulation - fourth edition*. Prentice Hall, 2005.

[2] Kenneth L. Calvert, M. B. Doar, and Ellen W. Zegura. Modeling internet topology. *IEEE Communication Magazine*, pages 160–163, 1997.

[3] Laura C. Carrington, Michael Laurenzano, Allan Snavely, Roy L. Campbell, and Larry P. Davis. How well can simple metrics represent the performance of HPC applications? In *Proc. of the 2005 ACM/IEEE conference on Supercomputing*, page 48, Washington, DC, USA, 2005. IEEE Computer Society.

[4] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc., 1991.

[5] Thomas Fahringer and Clovis Seragiotto. Automatic search for performance problems in parallel and distributed programs by using multi-experiment analysis. In Sartaj Sahni, Viktor K. Prasanna, and Uday Shukla, editors, *HiPC*, volume 2552 of *Lecture Notes in Computer Science*, pages 151–162. Springer, 2002.

[6] Sally Floyd and Vern Paxson. Difficulties in simulating the internet. *IEEE/ACM Trans. Netw.*, 9(4):392–403, 2001.

[7] Dan Geiger, Thomas Verma, and Judea Pearl. d-separation: From theorems to algorithms. In Max Henrion, Ross D. Shachter, Laveen N. Kanal, and John F. Lemmer, editors, *UAI*, pages 139–148. North-Holland, 1989.

[8] Anthony J. G. Hey, Alistair N. Dunlop, and Emilio Hernández. Realistic parallel performance estimation. *Parallel Computing*, 23(1-2):5–21, 1997.

[9] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing*. Benjamin/Cummings, Redwood City, CA, 1994.

[10] Jan Lemeire. *Learning Causal Models of Multivariate Systems and the Value of it for the Performance Modeling of Computer Programs*. PhD thesis, Vrije Universiteit Brussel, 2007.

[11] Jan Lemeire, Erik Dirkx, and Frederik Verbist. Causal analysis for performance modeling of computer programs. *Scientific Programming*, 15(3):121–136, 2007.

[12] Gabriel Marin and John Mellor-Crummey. Cross-architecture performance predictions for scientific applications using parameterized models. In *SIGMETRICS '04/Performance '04: Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 2–13, New York, NY, USA, 2004. ACM Press.

[13] Judea Pearl. *Causality. Models, Reasoning, and Inference*. Cambridge University Press, 2000.

[14] A. Snavely, N. Wolter, and L. Carrington. Modeling application performance by convolving machine signatures with application profiles. In *WWC '01: Proceedings of the Workload Characterization, 2001.*, pages 149–156, Washington, DC, USA, 2001. IEEE Computer Society.

[15] Allan Snavely, Laura Carrington, Nicole Wolter, Jesús Labarta, Rosa M. Badia, and Avi Purkayastha. A framework for performance modeling and prediction. In *SC*, pages 1–17, 2002.

[16] Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search*. Springer Verlag, 2nd edition, 1993.

[17] Peter Spirtes, Clark Glymour, Richard Scheines, and Joseph Ramsey. The TETRAD project. `http://www.phil.cmu.edu/projects/tetrad/`.

[18] Jin Tian and Judea Pearl. A general identification condition for causal effects. In *AAAI/IAAI*, pages 567–573, 2002.

[19] Hong Linh Truong and Thomas Fahringer. SCALEA: A performance analysis tool for distributed and parallel programs. In Burkhard Monien and Rainer Feldmann, editors, *Euro-Par*, volume 2400 of *Lecture Notes in Computer Science*, pages 75–85. Springer, 2002.

[20] M.P. Wand and M.C. Jones. *Kernel Smoothing*. Chapman & Hall, London, UK, 1995.

[21] Ellen W. Zegura, Kenneth L. Calvert, and Michael J. Donahoo. A quantitative comparison of graph-based models for Internet topology. *IEEE/ACM Transactions on Networking*, 5(6):770–783, 1997.