HAL
open science

# Process Affinity, Metrics and Impact on Performance: an Empirical Study

Cyril Bordage, Emmanuel Jeannot

## To cite this version:

Cyril Bordage, Emmanuel Jeannot. Process Affinity, Metrics and Impact on Performance: an Empirical Study. [Research Report] RR-9132, Inria Bordeaux Sud-Ouest. 2017. hal-01667273

**HAL Id: hal-01667273**

**https://inria.hal.science/hal-01667273v1**

Submitted on 19 Dec 2017

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Process Affinity, Metrics and Impact on Performance: an Empirical Study

Cyril Bordage, Emmanuel Jeannot

RESEARCH REPORT
N° 9132
Décembre 2017
Project-Team TADaaM

ISSN 0249-6399 ISRN INRIA/RR--9132--FR+ENG

# Process Affinity, Metrics and Impact on Performance: an Empirical Study

Cyril Bordage, Emmanuel Jeannot

Project-Team TADaaM

Research Report n° 9132 — Décembre 2017 — 20 pages

**Abstract:** Process placement, also called topology mapping, is a well-known strategy to improve parallel program execution by reducing the communication cost between processes. It requires two inputs: the topology of the target machine and a measure of the affinity between processes. In the literature, the dominant affinity measure is the communication matrix that describes the amount of communication between processes. The goal of this paper is to study the accuracy of the communication matrix as a measure of affinity. We have done an extensive set of tests with two fat-tree machines and a 3d-torus machine to evaluate several hypotheses that are often made in the literature and to discuss their validity. First, we check the correlation between algorithmic metrics and the performance of the application. Then, we check whether a good generic process placement algorithm never degrades performance. And finally, we see whether the structure of the communication matrix can be used to predict gain.

**Key-words:** process placement, topology mapping, MPI, communication, algorithm, communication modeling, performance metric

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

# Affinité entre les processus, métriques et impact sur les performances : étude expérimentale

**Résumé :** Le placement de processus en prenant en compte la topologie de la machine est une technique bien connue pour réduire le temps d'exécution d'un programme parallèle en diminuant le coût des communications entre les processus. Il nécessite deux entrées : la topologie de la machine cible, et une mesure de l'affinité entre les processus. Dans la littérature, la mesure d'affinité qui prédomine est la matrice de communication qui comptabilise les communications entre les processus. Le but de ce papier est d'étudier la pertinence de la matrice de communication comme mesure de l'affinité. Dans ce but, nous avons réalisé un grand nombre de tests sur une machine de type fat-tree ainsi que sur un tore 3d, afin d'évaluer plusieurs hypothèse qui se retrouvent souvent dans la littérature et de discuter de leur validité. Pour cela, d'abord nous vérifions la corrélation entre des métriques algorithmiques et la performance de l'application. Ensuite, nous contrôlons qu'un bon algorithme de placement n'implique jamais une dégradation des performances d'une application. Et finalement, nous étudions la structure de la matrice de communication dans le but de voir si elle peut être utilisée dans la prédiction du gain.

**Mots-clés :** placement de processus, MPI, communication, algorithme, modèle de communication, métrique de performance

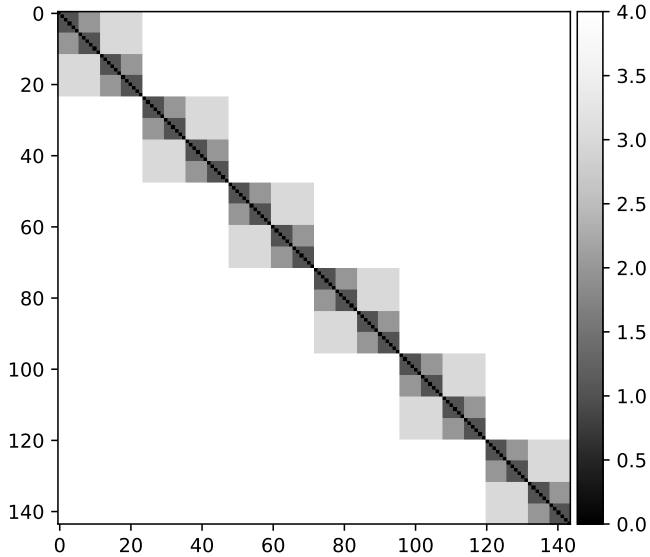



Figure 1: Distance matrix between cores in a fat-tree machine with 6 nodes with 24 cores

# 1 Introduction

We are currently seeing a deepening in the hierarchy of high-performance computing system. Nodes are composed of multicore processors with different levels of memory (standard DRAM, non-volatile memory, faster but smaller MCDRAM for KNL, etc.) and the network interconnecting these nodes can also be highly intricate with complex topology and high diameter. The consequence of these architectural features is that the performance of the parallel applications highly depends on the nodes allocated for the job as well as the mapping of these jobs. Process placement (also known as topology mapping) is an active field of research that deals with the development of strategies targeting the improvement of parallel applications by carefully allocating processes onto the resources [13]. The goal is to reduce the communication by mapping close to each other processes that communicate the most.

The communication time depends on the algorithm implemented in the application: it depends on the quantity of data to be exchanged. Moreover, since all computing resources are not directly connected, it also depends on the distance between the running processes as well as the speed of the different links. Figure 1 shows what can be the distances (in number of hops) between cores in a fat-tree machine with 6 nodes with 24 cores each (two processors made of two NUMA nodes with 6 cores each). We see clearly blocks of same distances.

Hence, it seems natural to put closer two processes that communicate a lot to reduce the communication cost. To this purpose, we need to adapt the execution of parallel applications to the target machine according to its specific topology.

To address this problem, process placement algorithms have been proposed. They use two input models: the target machine, often the topology graph, and the affinity between processes, often given as a communication graph, and they compute a mapping. Gains above 30% in terms of execution time have been reported in [14, 15, 6].

In this paper, we will not propose another process placement algorithm but we rather address generic questions about this research field. Here, we study how is defined and used affinity. This

paper therefore aims at studying the following questions. What are the limits of these models? Are performance correlated with metrics to optimize? What is a good measure of affinity? Indeed, such questions are often overlooked in the literature.

Our scientific method is the following. We formulate a hypothesis that is often made in the literature and we conduct experiments in order to check this hypothesis and try to define its limits.

The remaining of the paper is organized as follows. In Section 2, we present the context for our study. Then, in Section 3, we formulate three hypotheses about processes affinity, placement and metrics. We experimentally check these hypotheses in Section 4. Section 5 shows the related work. Finally, we present our conclusions in Section 6, and we discuss improvements and future work.

## 2 Context

We consider parallel applications composed of processing entities (e.g. processes) that exchange data through message passing (e.g. using MPI [11], Charm++ [16], etc.). In this case the total or a sub-part of the data exchanged is accountable for the execution walltime. Indeed, except the case of a full overlap between communication and computation, each message takes a time that depends at least on its size, the available bandwidth and the latency of the interconnection network (or of the memory, if the message is sent between processes on the same node). The mapping of the processing entities onto the computing resources consists in deciding where each of them will execute its computations [13]. Such mapping has been shown to improve the overall performance of the application by reducing the communication time [5, 17, 14]. The idea is that highly communicating processing entities should be mapped as close as possible onto the topology in order to reduce communication cost. Indeed, the communication cost of a message is increasing with the distance between the processing units hosting the sender and the receiver (see Fig. 1). Moreover, the bandwidth within a node is much larger than the network interconnect and each hop costs some time in terms of latency.

Algorithmically speaking the mapping problem takes two inputs [13]: a description of the machine enabling to compute the communication cost of the messages and a model of the application describing the affinity between the computing entities. The notion of affinity is important and encompasses the fact that some processes should be mapped close to each other. In most of the works in the literature, the affinity between processes is given by a communication matrix that describes the amount of communication between pairs of processes. Indeed, as we plan to reduce communication cost, the more communication a pair of processes has, the more affinity these processes have. There exist several ways of measuring the communication cost. In the context of distributed memory computation, some authors [8] use the amount of memory that is shared between threads. In the context of message passing, the standard way is to use either the number of messages, the total size of the exchange data or the average size of the messages [15].

In this paper, we only consider applications that have a reproducible communication graph (i.e. is the same from one run to another) or computable when required. Many applications provide reproducible communication patterns: dense linear algebra kernels (LU, QR, Cholesky), stencil code, etc. In this case, extracting the communication matrix from an application is done through monitoring. We run the application once and extract the communication pattern based on the messages exchanged between processes. In this work, we use a low-level monitoring tool inside the Open MPI implementation that has the unique advantage of being able to track messages of collective communication once such collectives have been decomposed in point-to-point communication [3]. In some other cases, the communication pattern is computable at launch

time or at runtime. For instance in Charm++, the runtime system provides the communication volume between chares. In many mesh-based applications, the communication pattern is driven by the domain decomposition. The mesh describing the model to be processed is partitioned in subdomains. Each subdomain is executed by a given process and the communication pattern between processes is directly derived from the subdomain graph as communications will be done through halo exchange between neighboring subdomains.

In order to evaluate a placement, several metrics have been proposed. Let $\sigma$ be the mapping function computed by a given algorithm, i.e. $\sigma(i)$ is the processing unit where process $i$ is placed. Let $C$ be the communication matrix, i.e. $C(i,j)$ is the amount of communication between processes $i$ and $j$. As we do not distinguish between sending and receiving messages we assume that $C$ is symmetric. Let $d$ be the distance function on the topology graph i.e. $d(a,b)$ is the number of hops between nodes $a$ and $b$. Let $B$ be the cost to transfer data between two nodes. The unit of $B$ must be coherent with the unit of $C$ (i.e. $B(\sigma(i),\sigma(j)) \times C(i,j)$ is the cost for moving the data between process $i$ and $j$ after the mapping). The literature proposes several mapping metrics that are the targets of the optimization (often minimization) problem:

- HopByte [19] is the accumulated cost of all the product between messages cost and the number of hops they have to traverse:

$$HB = \sum_{i,j} d(\sigma(i),\sigma(j))C(i,j)$$

- SumCom [18] is the sum of all the data movement cost:

$$SC = \sum_{i,j} B(\sigma(i),\sigma(j))C(i,j)$$

- MaxCom is derived from SumCom and is the maximum of all the data movement cost:

$$MC = \max_{i,j} B(\sigma(i),\sigma(j))C(i,j)$$

Another way of measuring the performance of the mapping is to simulate the communication inside the topology. Indeed, one drawback of the above metrics is that they do not take into account network contention and temporality. One way of partially addressing this issue is to simulate data flow on the network taking into account topology, routing, bandwidth and latency. In this work, we use the SimDag framework of SimGrid [4] where each process is a task with no computation cost and communication costs correspond to entries in $C$. Unfortunately, since we provide a communication matrix with no time information, we only express the overall communication, hence the contention could be overestimated.

Minimizing these metrics is an NP-complete problem (unless for special instances) as it is reducible from the graph embedding problem. Therefore, many heuristics and standard placement have been proposed. Round robin (RR) consists in having mapping processing entities $i$ on computing resource $i$ (hence $\sigma$ is the identity function). The goal of this work is not to compare the different strategies of the literature. We rather aim at evaluating the communication matrix model as a relevant affinity measure of applications. To do so, we evaluate the performance of the mapping with several angles. For mapping, we will take the TreeMatch (TM) strategy [15] developed by a subset of the authors and provide a comparison against RR and random (RND) mapping: the idea is to study if careful mapping provides significant improvement. We will also look at the correlation between metrics and performance: does minimizing mapping metrics leads

to performance improvement? We will also investigate the question of communication matrix characteristics and its impact on the benefit of mapping.

To perform this study we use the NAS parallel benchmarks (NPB) as they comprise an important and recognized set of typical parallel programs. However, these benchmarks are mainly computational ones and are not perfectly suited to our study. Indeed, mapping optimizations concern only communication. Therefore, if computation dominates the execution time, the gain (or loss) can be difficult to observe. Hence, in addition to NPB, we have chosen a MPI-based mini-application called miniGhost [2]. Its main advantage is the possibility to set many parameters that have an influence on the ratio of computation and communication (number of variables by stencil point, number of iterations, size of the problem, number of cells by process, etc.) and for which the structure of the execution can be tuned to see different behavior of mapping strategy (stencil dimension 2D or 3D, stencil size in each dimension, connectivity between stencil point, communication strategy). We also chose miniGhost as there is no overlap between communication and computation. Indeed, as we are trying to improve the communication time, such an overlap could distort the experiments and hence hinder one of our goal: finding correlation between the execution time gained with the mapping and other metrics. Having such a tunable application is a huge asset for this experimental study.

# 3 Hypotheses Formulation

Process placement consists in mapping processes onto resources in order to optimize execution time. This optimization is done based on some inputs and after an algorithmic process. There is lot of research in this area and designing new and efficient algorithms requires a deep understanding of the interaction between models and metrics.

Therefore, we formulate here hypotheses that are often made when designing process placement strategies and we give the intuitions that drive them. Our goal is to study these hypotheses in order to give hints to algorithm developers when they design and analyze their own solution.

A first set of questions concerns the metric to be optimized when computing the placement. As described in Section 2, some metrics are proposed in the literature. However, these metrics concern an evaluation of the mapping within the machine model and application affinity model (i.e. here the communication matrix). A good metric should be such that if it tells that a mapping is better, then the execution time should be lower. Even if it is not expected to have a quantitative gain (i.e. if the metric is $x$ time better, then the runtime is $x$ time lower), at least we should see a correlation between metrics and performance. We can therefore formulate a first hypothesis:

**Hypothesis A: algorithmic metrics are correlated with performance**

This hypothesis means that a mapping improving an algorithmic metric (e.g. HopByte, SumCom, etc.) should lead to a decrease of the execution time of the application. Testing hypothesis A can be done by computing the different metrics on two mappings (Round-Robin and Random) as well as measuring the corresponding runtimes. Moreover, such experiments can be used to evaluate each metric: the better the correlation is, the more relevant the metric is.

If we find high correlated metrics regards to performance, we can wonder if a placement optimizing these metrics never degrades performance compared to standard mapping (i.e. RR). We can therefore made the following hypothesis:

**Hypothesis B: a process placement algorithm that is optimized regards to relevant metrics never degrades performance**

Testing hypothesis B is not very difficult. We take a process placement optimizing the considered metric(s) and we compare the execution of the application with it and with the round-robin placement.

The last set of questions we want to address in this paper concerns the relationship between the communication matrix structure and the performance gain: does the communication matrix structure impacts performance gain? The idea is to characterize the communication pattern by a single value that computes the potential gain of an optimized mapping. For instance, if the non-zero values of the matrix are only near the diagonal, this means that process $i$ only communicates with processes $i-1$ and $i+1$. In this case the potential gain of a clever mapping is very low because the distance between consecutive indices of cores will be often the lowest possible. On the contrary, if the large values are far from the diagonal and the values have a huge variability then potential gain may be important. This leads to formulate the following hypothesis:

**Hypothesis C: communication matrix structure and values impact the performance gain**

To test hypothesis C we need to compute different matrix metrics from the literature and compare their value to the gain when executing a good placement. The hypothesis can be validated if we see a correlation between matrix metrics and gains.

# 4 Hypotheses Testing

To check our hypotheses, a set of tests were run on several machines. For each test, that is, each parameter set on each machine, we have executed ten times the same application with same inputs. We have run a t-test on the results to validate them and the ten runs have been compiled by computing the median value.

## 4.1 Experimental environment

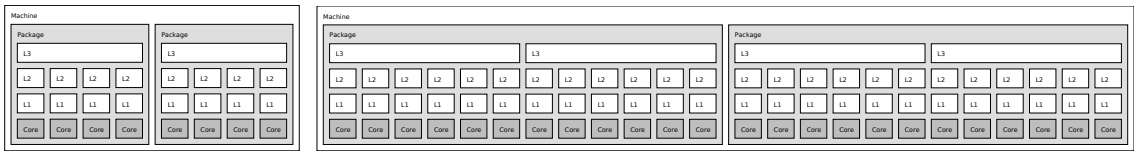
The first test machine is Plafrim 1, a 68 nodes machine with fat-tree network. It is an InfiniBand QDR network made of four leaf switches with around 17 nodes each. Each node has two quad-core Intel Xeon X5550 processors, as shown in Fig. 2a. Therefore, the topology of the whole machine, considering the network topology and the intra-node topology is a fat-tree which number of sons from the root is 4, 18, 2 and 4. Since a fat-tree is a balanced tree, we use 18 for the number of nodes by switch as it is the maximum.

The second machine is Plafrim 2, a 88 node machine with fat-tree network.It is an InfiniBand QDR network made of four leaf switches with 22 nodes each. Each node contains two Intel Xeon E5-2680 v3 processors (24 cores total, split in 4 NUMA nodes with 6 cores each), as shown in Fig. 2b. For the whole fat-tree, the number of sons from the root is 4, 22, 2, 2 and 6.

The third machine is Blue Waters, a Cray XE/XK hybrid machine composed of $22,640$ nodes with AMD 6276 Interlagos processors all connected by the Cray Gemini torus interconnect[1]. The topology is a $24 \times 24 \times 24$ Torus with a tree which number of sons from the root is 2, 2, 2, 4 and 2.

To generate the process mapping, Netloc, a tool included in hwloc [10], after discovering the topology of the machine, takes a communication matrix as input and uses a mapping tool like TreeMatch [15] to generate the topology-aware mapping with one process per core as a rank file

[1]This machine is used only for testing hypothesis A as our mapping algorithm is not designed for Torus topologies. Extensive study on that machine is left to future work




(a) Node of Plafrim 1 (b) Node of Plafrim 2

Figure 2: Node topolgoy of the test machines, from lstopo.

for MPI. The communication matrices are generated with a monitored version of Open MPI [9]. With each pair of processes, we get statistics about communication: number of messages, total number of bytes, and average size. For affinity measure, we used both the number of messages and the total size of the communication.

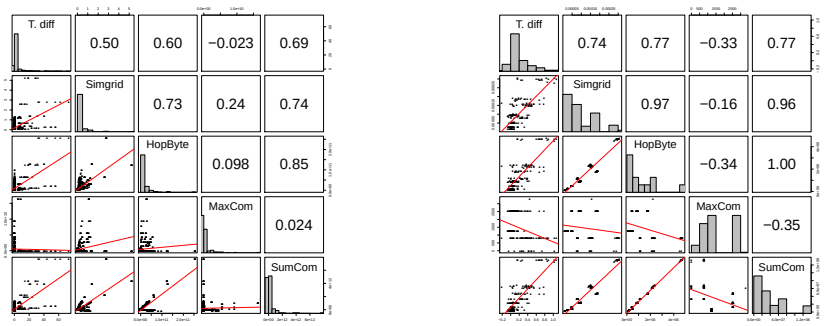
For the miniGhost parameters we explore several dimensions. We tried two communication method strategies: SVAF (data aggregated by face) and BSPMA BSP synchronous mode). The number of variables per stencil point is set to 20 or 40 and the dimension of the stencil is 24, 48 or 96 in each dimension (27 combinations). The tests were done with three stencil types as defined by miniGhost: 21, 23 and 24. We tested execution on 4 and 8 nodes for Plafrim 1, on 1, 2, 3 and 6 nodes Plafrim 2, and on 1, 2, 3, 4, 8, 16 on Blue Waters.

For the NAS benchmark, we have used the bt, cg, ft, lu and mg kernel with class form A to D (with only the largest number of processors in the later case).

## 4.2 Hypothesis A: algorithmic metrics are correlated with performance

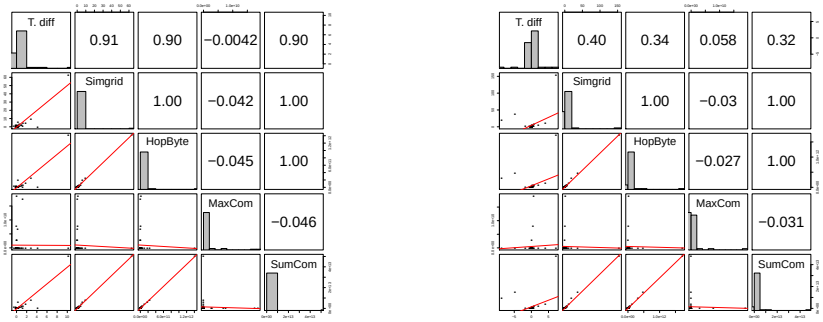
To check if we have algorithmic metric correlated with performance, we have computed the correlation between two sets of values: the gain in execution time and the difference between the metrics for two mappings (RR and RND) computed with size as an affinity measure. We do not use any special algorithmic process to improve the mapping in this section as we only want to see the impact of the mapping change on the performance and the metrics.

We use the difference between the different measurements because values can be very large and hence the difference between two placements can be important while ratio can be small.




(a) On Plafrim 1 (b) On Plafrim 2

Figure 3: Metrics correlation for the MiniGhost application. Difference between round-robin and random placement




(a) On Plafrim 1 (b) On Plafrim 2

Figure 4: Metrics correlation for the NAS benchmarks. Difference between round-robin and random placement

The results presented in Fig 3, show the correlation of our placement compared to the four performance metrics presented in

Sec. 2 for the miniGhost application and the Plafrim machines. Fig 4 present the same results but for the NAS benchmarks.

Results are read as follows. On the diagonal we have the different metrics and the execution time as well as the distribution of the difference between the RND mapping and the RR mapping. Under the diagonal we have the different projections of the different runs in the subspace defined by the metric on the line and the one on the column as well as the interpolation line. Above the diagonal, we have the correlation coefficient. This coefficient is between -1 (perfect anti correlation) and 1 (perfect correlation).

On both machines, the worst correlation is for the MaxCom metrics. This metrics fails to express the execution time gain and optimize it does not lead to improvement in general. The other metrics (Simgrid, SumCom and HopByte) perform similarly. In other experiments (where we compare TreeMatch and RR), we have seen that Simgrid has a better correlation with performance gain, than the two other metrics. This is not surprising, since Simgrid is the algorithm that will exploit the best the input matrix and the architecture topology by simulating communication.

Then, the correlations for HopByte and SumCom are of the same order. It is expected as HopByte is like SumCom with particular values for bandwidth. However, tuning the costs of communication for SumCom is a strong disadvantage. This tuning has to be done for one architecture but we need several applications to be sure it is not dependent of the application. However, this is out of the scope of this paper and will be done in future work.

Due to lack of space, we do not the show Figure for Blue Waters. Yet, on this machine, for HopByte and SumCom the correlations are high: 0.81. As for the fat trees, MaxCom shows a low correlation (0.2). However, with Simgrid, we have to investigate further as we see a low correlation (0.21) but with no explanation.

To conclude, we can say that there is a quite good correlation between the time we gain by other placements and our algorithm metrics, on fat-tree machines and on a 3d-torus machine. However, the HopByte metric has a strong advantage over its counterpart. Indeed, it does not required to be tuned according to the target machine: it considers only the topology of the machine and not the performance of its network. We believe this to be a decisive advantage, as gathering such information is error-prone, might be incomplete and subject to inaccuracy.

### 4.3 Hypothesis B: a process placement algorithm that is optimized regards to relevant metrics never degrades performance

As we have seen with the previous hypothesis, HopByte is well correlated with the performance and therefore seems to be a good candidate as a metric to be optimized in a process placement algorithm. To check the validity of hypothesis B with HopByte, we choose TreeMatch as the process placement algorithm.

TreeMatch [15] computes the mapping from the topology of the underlying machine and the behaviour of the application and optimizes the HopByte metric. It has shown good placement strategies for NAS applications. It takes as input a tree topology (where the leaves stand for computing resources and internal nodes correspond to switches or cache levels) and a matrix describing the affinity graph between processes. A hierarchy is extracted from the communication pattern that matches the topology tree hierarchy. The outcome is therefore a mapping of the processes onto the underlying computing resources. TreeMatch works only on fat-trees and we do not test this hypothesis on the 3d-torus machine.

To test hypothesis B, we first build the communication matrix by using the monitoring in Open MPI. Then, we compute an optimized placement with our TreeMatch algorithm, run our application with this mapping and compare the execution time against the time measured with RR mapping. We build two different types of communication matrices. The first contains the number of messages shared by processes and is called *msg*. The second contains the total size of data shared by processes and is called *size*. We carry out the experiments with these two matrices and compare them. For the miniGhost application results are displayed as empirical cumulative distribution functions (ECDF) in Fig. 5 and in Fig. 6 for the NAS benchmarks.

For the NAS benchmarks and miniGhost, on both machines, the best results are with the communication matrix using the size of the data. This shows that all the messages between processes have not the same size, and therefore the number of messages is not sufficient to model the affinity.

If we only consider results for size matrix, our mapping leads to performance improvement in general but it is not better for all cases. For minighost and for Plafrim 1, 83.3% show an improvement of performance, while for Plafrim 2, 69.1%. We have a median gain of 3.2% on one machine and 1.5% for the other one. It leads to up to 22% improvement while never degrading by more than 7% in the worst cases, depending on the machine. For the NAS benchmarks results are less impressive. On Plafrim 1 (resp. Plafrim 2) 75.9% (resp. 64.5%) of the cases are improved by using TreeMatch. The median gain is 0.33% on Plafrim 1 and 0.16% on Plafrim 2. The fact that minighost improvements are better than the NAS one is explained by the fact that the amount of communication is lower in NAS case and hence the potential gain is greater for the miniGhost application.

In conclusion, we can make two statements. First, the way we measure affinity (e.g. number of messages vs. size of the messages) impacts the way the mapping is computed and the overall performance. In pour case, using the size is a better measure if *affinity* however, this is application dependent and should be tested for each application. Secondly, the mapping algorithm is not beneficial in all cases, the result depends on the parameters of miniGhost. Hence, strictly speaking, hypothesis B is rejected but, in general, process mapping is beneficial.

### 4.4 Hypothesis C: communication matrix structure and values impact the performance gain

When we look at the communication matrix, it is difficult to evaluate a mapping. In Figure 7, for three different parameter sets, the communication matrices are shown for two mappings and

the gain represents the time saved (or lost) in percentage by using the mapping on the right vs the mapping in the left. The communication matrix on the right side is simply a permutation of the matrix on the left side. The permutation vector represents the changes of the new mapping as well as the changes in the communication matrix.

Interestingly, if we just look at the structure of the matrix, it seems quite difficult to predict if we will save time or not and it looks even more difficult to predict the value of the gain. However, if looking at the structure is not sufficient, then the question of designing matrix statistic (i.e. a single measure computed from the matrix values) that show the impact on the gain is a relevant question.

Several matrix statistics have been proposed in the literature [8, 7]. In [8], the authors studied affinity in the context of thread placement and NUMA architectures. They study if, given the communication matrix, it is possible to expect a potential gain when we remap threads by looking at the structure of the matrix. They propose a statistic (called *communication heterogeneity* – CH) that takes a communication matrix as input and compute a measure such that the higher its value the better they expect a gain from a careful mapping. The intuition is: *to benefit from thread mapping, it is necessary to have groups of threads that share the same data among each other, and not with other threads.* And conversely if all the threads share the same amount of data, the expected benefit is very low. Formally, they compute $M$ the communication matrix where all values of $M$ are normalized by the largest value of the original communication matrix $C$: $M[i][j] = C[i][j]/\max(C)$. Then CH, is the average variance of the communication of each thread. Let $n$ be the order of $M$:

$$CH = \frac{\sum_i \sum_j \left( \frac{\sum_k M(i,k)}{n} - M(i,j) \right)^2}{n^2}$$

The larger CH the larger the variance of the communication cost between processing entities (thread/processes). Hence, CH tends to measure the communication heterogeneity.

A subset of these authors have proposed, in [7], another statistic to measure if balancing communication (communication balance – CB) among threads could be beneficial. The idea is to measure how the average communication cost of each thread is different from the largest thread communication cost. Let $T(i)$ be the communication cost of a thread: $T(i) = \sum_j C(i,j)$. Then

$$CB = 1 - \frac{\sum_i T(i)/n}{\max(T)}$$

If $CB = 0$ this means that all the thread communication costs are equal and balanced while higher values indicate more imbalance within the thread communication costs.

The two above statistics were designed for a context that is slightly different from the one of this paper: shared memory thread placement based on data sharing. Here, we target process placement based on communication issued from message passing. Hence, we have also designed three original statistics to measure the potential benefit for process mapping based on the communication matrix.

The first statistic, called communication centrality (CC), measures how the communication is dispersed from the diagonal. The idea is that, if all the communication is performed around the diagonal of the communication matrix, then there is no opportunity for mapping (each process $j$ more or less communicates with process $j-1$ and $j+1$). For each line $i$, we compute $j_1 \geq 0$ and $j_2 \leq n$ such that half of the communication cost is between $C(i, j_1)$ and $C(i, j_2)$ and $i - j_1 = j_2 - i$. Then, we compute $R_i = (j_2 - j_1)/n$ the communication centrality of process $i$ ($R_i = 0$ means that

half the communication is on the diagonal and larger values means that more communication is off-diagonal). And $CC$ is the average value of $R_i$:

$$CC = \frac{1}{n} \sum_i R_i$$

The second statistic called neighbor communication fraction (NBC) follows the similar idea as CC. It computes the fraction of communication that is performed by neighboring processes. For each process $i$, we compute the fraction communication cost that is performed with their neighbor:

$$NBC^2 = 1 - \frac{\sum_i (C(i, i-1) + C(i, i+1))}{\sum_i \sum_j C(i, j)}$$

If all the communication is performed within neighbors then $NBC = 0$. Hence, the higher NBC, the more opportunity for mapping.

The last proposed statistic is called split fraction (SP(k)). It takes a parameter $k$ and computes the amount of communication that is done around block of $k \times k$ processes. This is useful to identify communication that are already performed within a node of $k$ cores:

$$SP(k) = 1 - \frac{\sum_s^{n/k-1} \sum_{l=0}^{k} \sum_{m=0}^{k} C(s \times k + l, s \times k + m)}{\sum_i \sum_j C(i, j)}$$

If all the communication is within these blocks then $SP = 0$ meaning that all communication is local within a node of $k$ cores. On Plafrim 1 we have taken $k = 8$ while on Plafrim 2 we have taken $k = 24$.

In Fig. 8 we plot the correlation graph between the different matrix statistics described, and the time difference between the RR mapping and the TreeMatch mapping for the miniGhost application. We plot the same correlation for the NAS benchmarks in Fig. 9. We compare with the ratio between the RR and TM as, contrary to the miniGhost case, timing difference are much lower and hence results are more visible with ratio of timings. The results are read as follows. On the diagonal of each pair plot, we have the different measure that are compared (time comparison, CH, CB, etc.). Moreover, in the same box we plot the histogram of the executions. On the upper part, we plot the correlations between each statistic (e.g. on Plafrim 1, the correlation between CB and the time difference is 0.54). On the above part, we plot the point in the dimension of both statistics as well as the linear regression line (in red). We can draw several interesting conclusions from these graphs.

First CB and CH are anti-correlated. This is counter-intuitive as, as stated above, they are designed to describe a similar expectation: the larger they are the better should be the benefit from a careful mapping. However, by carefully looking at the formula of CH and CB, we see that CH is proportional to the variance of the values of the communication matrix while CB is inversely proportional to the mean. However, in the communication matrix we are studying, these two measures (mean and variance) are correlated. Indeed, if we closely look at the values of the communication matrices we typically see that each line is composed of many zeros (each computing entities communicates with few other entities) and a set of similar large values (the processing entities roughly exchange same amount of data to communicating ones). Even if the non-zeroes values are sometimes different (see matrices in Fig. 7), modeling the distribution of the values of one line by a Bernoulli distribution is a very good fit. For instance, on the matrices of Fig. 7, we have taken the first line of each of them and computed $p$ the fraction of non-zero

[2] We of course, ensure $i - 1 \geq 0$ and $i + 1 \leq n$

values. Then, we have executed the Kolmogorov-Smirnov test of all the other lines against a randomly generated set of value following the Bernoulli distribution of parameter $p$ (scaled by the largest value of the line). In this case, the Kolmogorov-Smirnov test is used to determine if the hypothesis that the lines follow a Bernoulli distribution is correct. Yet, the statistic test has an average value of 0.15 and the average p-value is 0.95 validating this hypothesis. However, the mean of a Bernoulli random variable of parameter $p$ is $p$ and its variance is $p(1-p)$. For our case (values of $p < 0.5$), $p$ and $p(1-p)$ are highly correlated (the correlation coefficient is larger than 0.96). Hence, the mean and the variance are highly correlated. This explains why CB and CH are anti-correlated.

Secondly, the different statistics span different intervals. The values of CH are between 0 and 0.09, the values of CB are between 0.92 and 1 which are very small intervals and make values hard to distinguish. On the opposite, CC is between 0.03 and 0.47, NBC between 0.3 and 0.9 and SP between 0.1 and 0.9. These larger intervals help to make a better difference between the cases.

Thirdly, if we look at the correlations themselves, we see that, for minighost, on Plafrim 1 the best correlation is CB followed by SP[8] while on Plafrim 2 the best correlation is CC followed by SP[24]. However, on Plafrim 1 there is no correlation between performance gain and the CC statistic (the coefficient is -0.028) and on Plafrim 2 the correlation between performance and the CB statistic is only 0.24. Therefore, it appears that the most consistent statistic to predict gain is the SP. On the NAS benchmarks results are slightly different and the correlation are in general smaller. This is due to the fact that gain are in general smaller with the NAS benchmarks than with miniGhost as the amount of communication is larger for miniGhost than for the NAS. Again a consistent metric is the SP one (as well as NBC, which also determines the communication made to neighboring processes). This means that, taking into account the intra-node communication versus inter-node one is an insightful way of determining the potential benefit of a mapping.

To conclude on this hypothesis, it is not necessarily immediate to see, by looking at the communication pattern if potential gain through process mapping is achievable. However, we observe that there is a good and consistent correlation between the gain in execution time and the amount of inter-node communication on the target machine. We therefore see that the impact of matrix structure and values on the gain can be measured.

# 5 Related Work

Topology mapping is a very active research field. A survey on this subject ranging from models to heuristics and implementation is done in [13].

We have used the TreeMatch algorithm [15] to test the mapping. However, this work is not a comparison about the performance of this specific heuristic. Concerning process placement, several other approaches exist using graph embedding techniques [14], geometric partitioning based on application structure [6] or by internally exploiting the knowledge of the application communication pattern [1].

Aside from process placement, thread placement is a similar problem (where to map threads on cores) but with different inputs. In process placement, the communication between processes is used to measure affinity. In thread placement, the situation is different. The affinity between threads impacts the performance (if two threads share data then they should be mapped closed to each-other) or the data access impacts performance (threads should be mapped close to the memory bank where the pages they access are allocated). The second approach looks more promising as many result use it [8, 7, 12]. However, if the affinity approach is different the

algorithmic process is similar: it consists in minimizing the implicit communication cost of the application.

# 6 Conclusion

Process placement is a very active and important field of research. When designing algorithm researchers often make implicit hypotheses about models, parameters driving the application performance or target metrics. Surprisingly, these hypotheses have not been extensively tested. We think that questioning hypotheses in order to validate or invalidate them and define there limits is as important as designing the strategies themselves.

To do so, we have formulated, three hypotheses, provided a protocol to test them, perform experiments on two kind of applications (minighost and NAS benchmark) and discuss the results.

After testing our hypotheses, we can make the following statements. First, the type of applications (NAS vs. miniGhost i.e. high- vs low computation/communication ratio) impacts the results: a low computation/communication ratio is in general less discriminant than a higher one. Moreover, some algorithmic metrics (e.g. HopByte, SumCom and Simgrid) to optimize are generally correlated with the performance. However, these correlations are not always very high. A mapping algorithm that optimize these relevant metrics, can sometimes lead to a decrease in performance. Even if, most of the times, we have a performance gain, with some parameter sets, we see degradations. Furthermore, the algorithmic metric used in the placement algorithm and that exploits the communication matrix and the architecture graph can show limitations in some cases. Last, in general, using the structure of the communication matrix as an impact on the performance gain when optimizing the placement. Interestingly, on fat-trees the best matrix statistic is the one that needs to get input from the number of cores per socket. It is clear that other phenomenons, such as cache effects, contentions, message size distributions, etc. have to be taken into account to describe performance more accurately.

This work has some limitations. Hence, it would be interesting to extend this study towards several directions. First, hypotheses have only been tested on few nodes. Tests on more nodes, with more levels of switches or with more hops, could be relevant since often applications need a lot of nodes. With that, a study on how the number of nodes influences the correlations could be a good further work. In addition, we want to extend it on other architecture and topologies. Then, using the communication matrix as input for placement algorithm has some limitations. It does not include the time dimension and it can be restricting for applications having several phases during their execution. Moreover, such model makes difficult to estimate correctly communication contention. Also, we do not take account of the distribution of the sizes of the messages in the communication. In future work, we intend to refine the communication matrix model to improve model consistency.

# Acknowledgment

This work is partially funded under the ITEA3 COLOC project #13024. Experiments presented in this paper were carried out using the PLAFRIM experimental testbed, being developed under the Inria PlaFRIM development action with support from Bordeaux INP, LABRI and IMB and other entities: Conseil Régional d'Aquitaine, Université de Bordeaux and CNRS and ANR in accordance to the programme d'investissements d'Avenir (see `https://www.plafrim.fr/`). We thank the JLESC structure `https://jlesc.github.io`) for providing us with some hours on the BlueWaters machine. We also want to thank Clément Foyer for providing us with the Open MPI

monitoring framework and Brice Goglin and Louis-Claude Canon for valuable comments about this work.

# References

[1] Hasan Metin Aktulga, Chao Yang, Esmond G Ng, Pieter Maris, and James P Vary. Improving the scalability of a symmetric iterative eigensolver for multi-core platforms. *Concurrency and Computation: Practice and Experience*, 26(16):2631–2651, 2014.

[2] Richard F Barrett, Courtenay T Vaughan, and Michael A Heroux. Minighost: a miniapp for exploring boundary exchange strategies using stencil computations in scientific parallel computing. *Sandia National Laboratories, Tech. Rep. SAND2011-5294832*, 2011.

[3] George Bosilca, Clément Foyer, Emmanuel Jeannot, Guillaume Mercier, and Guillaume Papauré. Online dynamic monitoring of mpi communication. In *23rd International European Conference on Parallel and Distributed Computing (EuroPar)*, page 12, Santiago de Compostella, aug 2017. Extended version in `https://hal.inria.fr/hal-01485243`.

[4] Henri Casanova, Arnaud Legrand, and Martin Quinson. Simgrid: A generic framework for large-scale distributed experiments. In *Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on*, pages 126–131. IEEE, 2008.

[5] Hu Chen, Wenguang Chen, Jian Huang, Bob Robert, and Harold Kuhn. Mpipp: an automatic profile-guided parallel process placement toolset for smp clusters and multiclusters. In *Proceedings of the 20th annual international conference on Supercomputing*, pages 353–360. ACM, 2006.

[6] Mehmet Deveci, Sivasankaran Rajamanickam, Vitus J Leung, Kevin Pedretti, Stephen L Olivier, David P Bunde, Umit V Catalyurek, and Karen Devine. Exploiting geometric partitioning in task mapping for parallel computers. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 27–36. IEEE, 2014.

[7] Matthias Diener, Eduardo HM Cruz, Marco AZ Alves, Mohammad S Alhakeem, Philippe OA Navaux, and Hans-Ulrich Heiß. Locality and balance for communication-aware thread mapping in multicore systems. In *European Conference on Parallel Processing*, pages 196–208. Springer, 2015.

[8] Matthias Diener, Eduardo HM Cruz, Laércio L Pilla, Fabrice Dupros, and Philippe OA Navaux. Characterizing communication and page usage of parallel applications for thread and data mapping. *Performance Evaluation*, 88:18–36, 2015.

[9] Edgar Gabriel, Graham E Fagg, George Bosilca, Thara Angskun, Jack J Dongarra, Jeffrey M Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, et al. Open mpi: Goals, concept, and design of a next generation mpi implementation. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pages 97–104. Springer, 2004.

[10] Brice Goglin. Managing the topology of heterogeneous cluster nodes with hardware locality (hwloc). In *High Performance Computing & Simulation (HPCS), 2014 International Conference on*, pages 74–81. IEEE, 2014.

[11] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: portable parallel programming with the message-passing interface*, volume 1. MIT press, 1999.

[12] Jens Gustedt, Emmanuel Jeannot, and Farouk Mansouri. Optimizing locality by topology-aware placement for a task based programming model. In *Cluster Computing (CLUSTER), 2016 IEEE International Conference on*, pages 164–165. IEEE, 2016.

[13] Torsten Hoefler, Emmanuel Jeannot, and Guillaume Mercier. An overview of process mapping techniques and algorithms in high-performance computing, 2014.

[14] Torsten Hoefler and Marc Snir. Generic topology mapping strategies for large-scale parallel architectures. In *Proceedings of the international conference on Supercomputing*, pages 75–84. ACM, 2011.

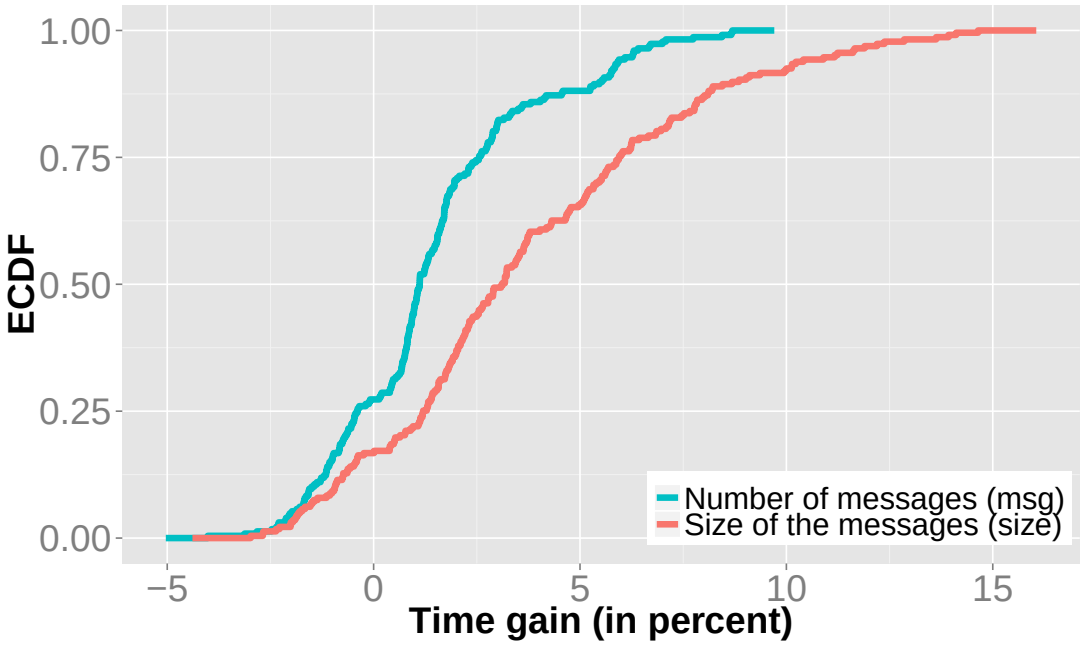
[15] Emmanuel Jeannot, Guillaume Mercier, and François Tessier. Process Placement in Multicore Clusters: Algorithmic Issues and Practical Techniques. *IEEE Transactions on Parallel and Distributed Systems*, 25(4):993–1002, April 2014.

[16] Laxmikant V Kale and Sanjeev Krishnan. Charm++: a portable concurrent object oriented system based on c++. In *ACM Sigplan Notices*, volume 28, pages 91–108. ACM, 1993.

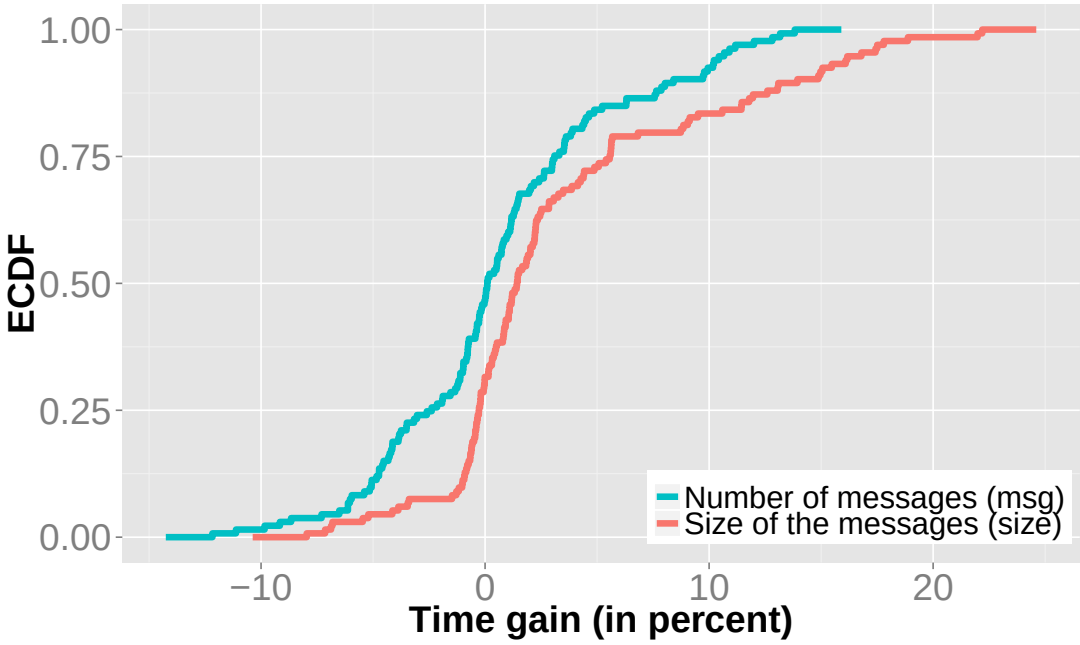
[17] Guillaume Mercier and Jérôme Clet-Ortega. Towards an efficient process placement policy for mpi applications in multicore environments. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pages 104–115. Springer, 2009.

[18] Jesper Larsson Traff. Implementing the mpi process topology mechanism. In *Supercomputing, ACM/IEEE 2002 Conference*, pages 28–28. IEEE, 2002.

[19] Hao Yu, I-Hsin Chung, Jose Moreira, et al. Topology mapping for blue gene/l supercomputer. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 116. ACM, 2006.

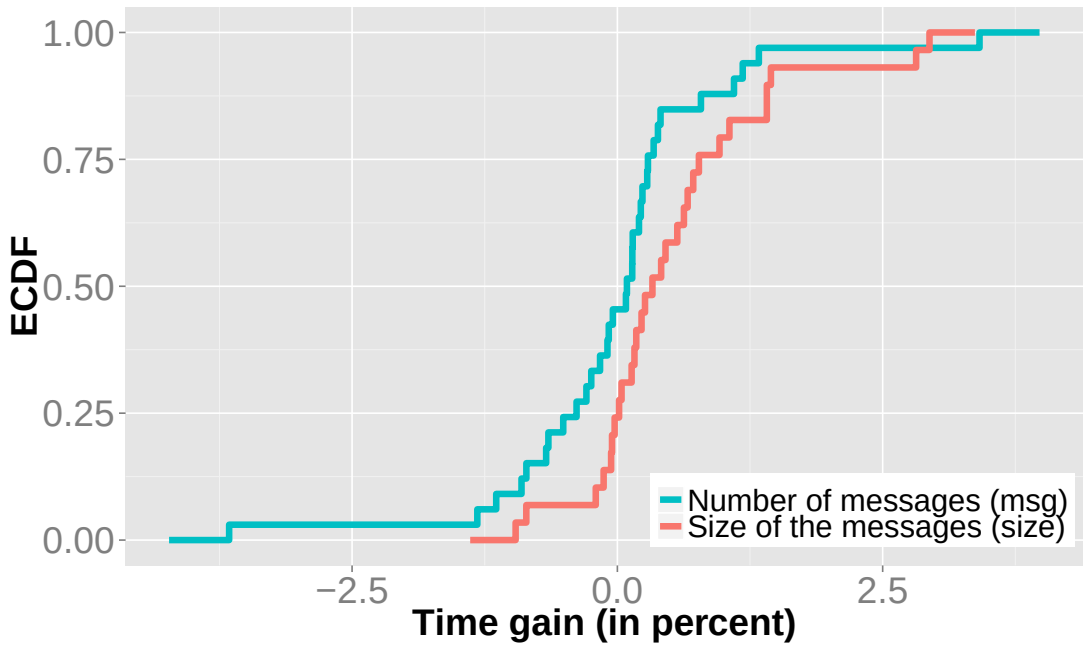



(a) On Plafrim 1

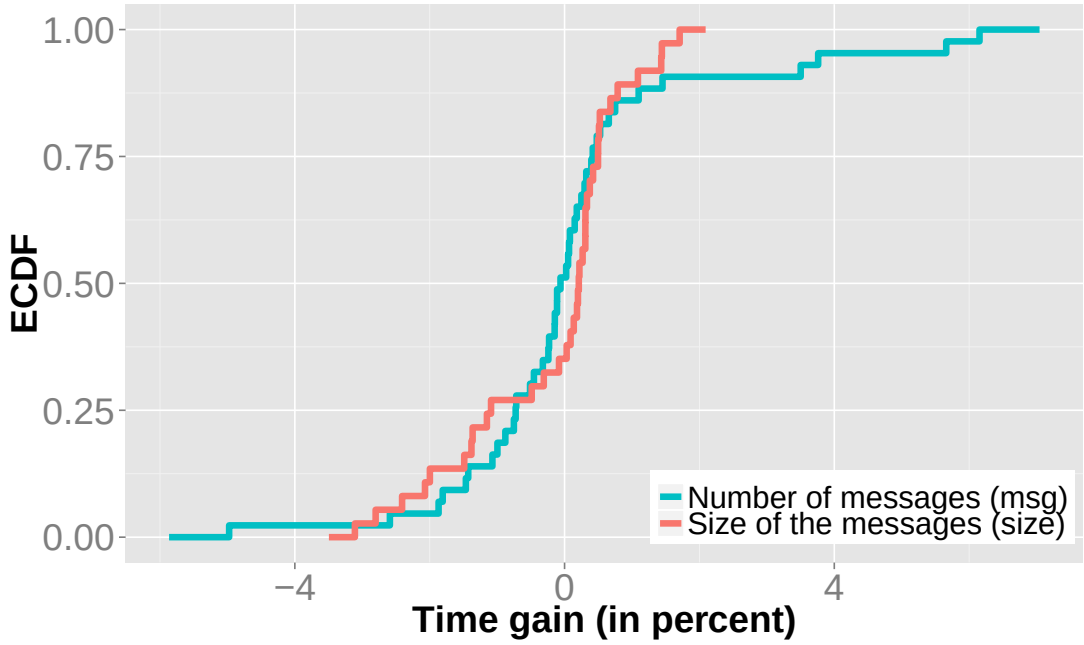



(b) On Plafrim 2

Figure 5: Average gain of TreeMatch against Round Robin for different miniGhost runs depending on the type of communication matrix

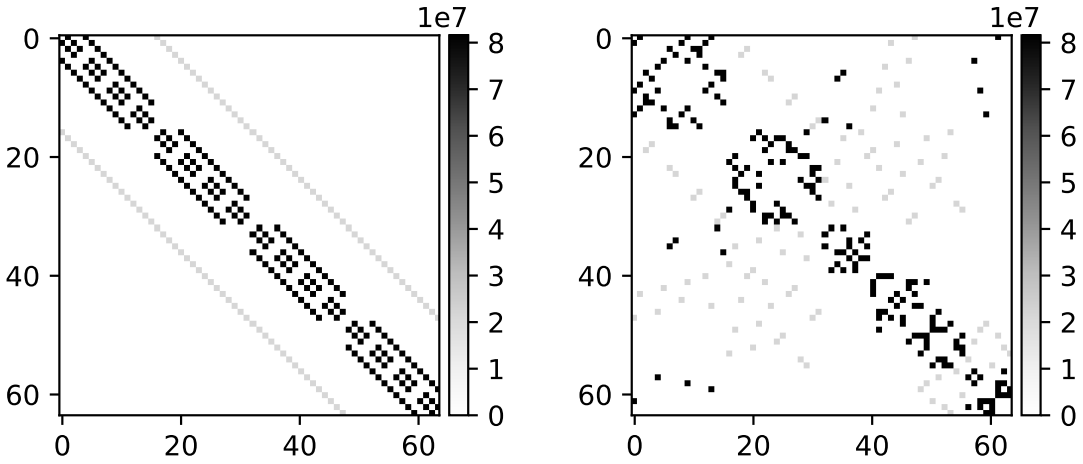



(a) On Plafrim 1

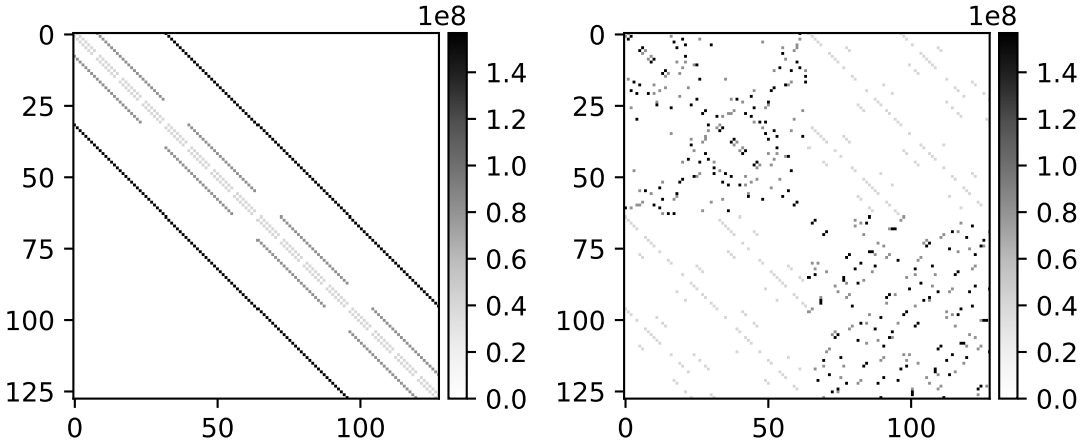



(b) On Plafrim 2

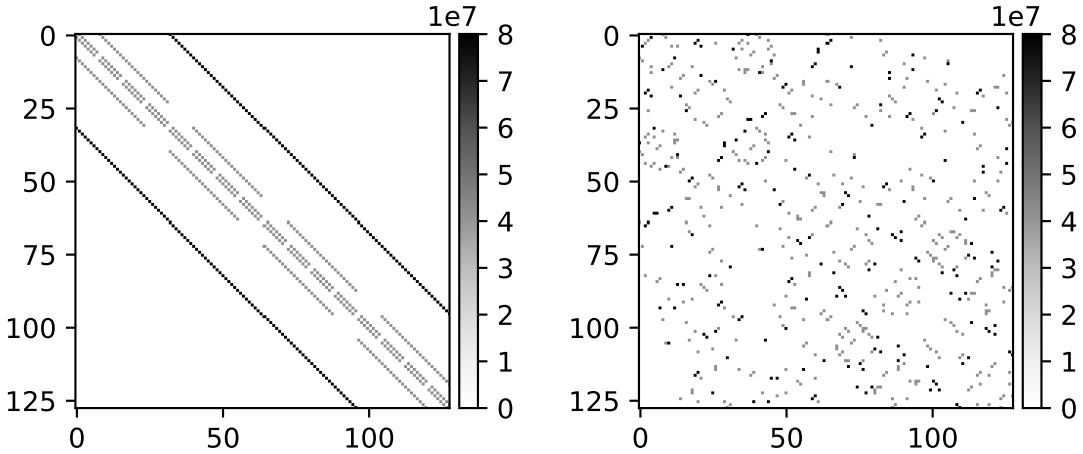
Figure 6: Average gain of TreeMatch against Round Robin for different NAS runs depending on the type of communication matrix




(a) Gain with the permutation on the right side: -5.4%



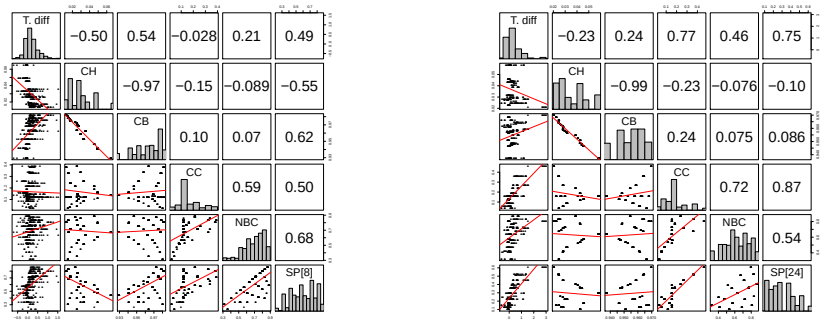

(b) Gain with the permutation on the right side: 20.8%




(c) Gain with the permutation on the right side: 24.6%

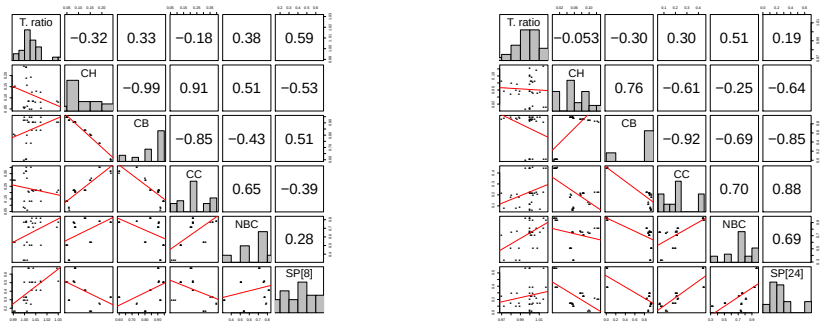


Figure 7: Communication matrices depending on the mapping for three different parameter sets.




(a) On Plafrim 1 (b) On Plafrim 2

Figure 8: Matrix statistic correlation for the minighost applicationwith the performance difference between TM and RR




(a) On Plafrim 1 (b) On Plafrim 2

Figure 9: Matrix statistic correlation for the NAS benchmarks with the performance ratio between TM and RR

**RESEARCH CENTRE**
**BORDEAUX – SUD-OUEST**

200 avenue de la Vieille Tour
33405 Talence Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399