HAL
open science

Predicting Performance of Communications and Computations under Memory Contention in Distributed HPC Systems

Alexandre Denis, Emmanuel Jeannot, Philippe Swartvagher

To cite this version:

Alexandre Denis, Emmanuel Jeannot, Philippe Swartvagher. Predicting Performance of Communications and Computations under Memory Contention in Distributed HPC Systems. International Journal of Networking and Computing, 2023, Special Issue on Workshop on Advances in Parallel and Distributed Computational Models 2022, 13 (1), pp.30. hal-03871630

HAL Id: hal-03871630

https://inria.hal.science/hal-03871630v1

Submitted on 9 Jan 2023

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.

International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 13, Number 1, pages 1–30, January 2023

Predicting Performance of Communications and Computations under Memory Contention in Distributed HPC Systems

Alexandre Denis, Emmanuel Jeannot, Philippe Swartvagher
Inria Bordeaux – Sud-Ouest
Bordeaux, France

Received: (received date)
Revised: (revised date)
Accepted: (accepted date)
Communicated by Editor's name

Abstract

To amortize the cost of MPI communications, distributed parallel HPC applications can overlap network communications with computations in the hope that it improves global application performance. When using this technique, both computations and communications are running at the same time. But computation usually also performs some data movements. Since data for computations and for communications use the same memory system, memory contention may occur when computations are memory-bound and large messages are transmitted through the network at the same time.

In this paper we propose a model to predict memory bandwidth for computations and for communications when they are executed side by side, according to data locality and taking contention into account. Elaboration of the model allowed to better understand locations of bottleneck in the memory system and what are the strategies of the memory system in case of contention. The model was evaluated on many platforms with different characteristics, and showed a prediction error in average lower than 4 %.

Keywords: HPC, MPI, Memory Contention, NUMA, Bandwidth, Predictive Models, Multicore Processing

1 Introduction

The key to reach a good scalability in distributed high-performance computing (HPC) is to reduce the cost of communications. One way of amortizing this cost as done by modern HPC applications consists in running computation alongside communications. This technique is known as communication and computation overlap, which consists in running the communications in background, while the computation is performed, in the hope that their cost becomes basically free.

However, computation moves data between memory and cores. When overlapping communications and computation, data movement for the computation and for the network may share parts of the path in the machine memory system. Contention can occur on this path, between these two kinds of streams. Figure 1 illustrates an example of such a situation: computing cores use data stored on a specific NUMA (*Non-Uniform Memory Access*) node and, in this case, communications store data received from the network on the same NUMA node. Both data streams travel through same paths of the memory system: contention can occur in the memory controller of a processor, the

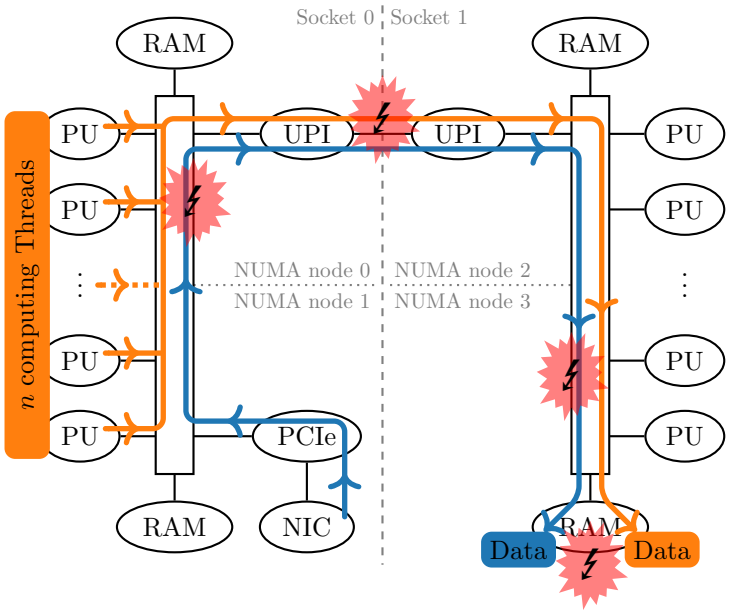


Figure 1: Contention between computations and communications can occur at different locations. The *inter-socket bus* is called *Infinity Fabric* (IF) on AMD processors, and *Ultra Path Interconnect* (UPI) on Intel processors.

inter-socket connection link or the controller of a NUMA node. These bottlenecks can reduce performance of both computations and communications, while computations/communications overlap is usually set up to save execution time.

We observed [8] that contention between computations and communications actually happens in practice, and we have shown that several factors can impact it: data placement, message size and arithmetic intensity of computing kernels. Performance is the most reduced when computing kernels are memory-intensive (putting important pressure on memory buses), big messages are exchanged (thus moving big messages through memory buses) and data to send to the network is located on a NUMA node different than the one where the network interface is plugged to.

In this paper, we propose a model of this contention between computations and communications. Given a number of computing cores, the model can predict memory bandwidth available for computations and communications, when they are executed simultaneously, while taking into account the locality of data they manipulate. More than just predicting performance, the proposed model allows us to test our hypotheses about the internal working of processors' memory system, how they deal with contention between different kinds of streams. This paper extends an article published in the 24th *Workshop on Advances in Parallel and Distributed Computational Models* [9]. This extended version explains the model in a more incremental way to justify all its elements, presents experimental results on more machines and include two appendices presenting algorithmic versions of the equations defining the model and values of model parameters obtained on each machine.

The rest of the paper is organized as follows: section 2 introduces context and initial hypotheses to build the model. Then, section 3 explains the model. Model predictions are evaluated and discussed in section 4. Finally section 5 presents related work and section 6 summarizes our findings.

2 Context and hypotheses

Since different kinds of data streams share the same memory bus, it is possible to sum the measured bandwidths of each data stream, to get the overall occupancy of the bus capacity, from a bandwidth point of view, like Majo and Gross did [14]. Indeed, this assumption is the cornerstone of our model; once the bandwidth capacity of the bus is known, one has to distribute the available bandwidth

between computations and communications.

However, it is important to note that behaviours of processors and memory controllers regarding contention are not publicly documented by processor manufacturers. Moreover, the values they use to characterize hardware features (the memory controller bandwidth or the SMP (*Symmetric multiprocessing*) interconnect rate, for instance) can hardly be linked to experimental observations, nor directly used as parameters of the model.

Thus, we propose a model whose parameters are determined through experiments rather than theoretical capacity of hardware. We make our own set of hypotheses explaining memory bandwidth in case of contention, as well as our own set of benchmarks to get model parameters.

2.1 Contention behaviour

Memory buses have a finite bandwidth. When this capacity (or threshold) is reached, the bus capacity is shared between all components accessing it, reducing memory bandwidth available for each accessor. Memory requests issued by CPU cores may have a different (often higher) priority than requests coming from PCIe devices, *e.g.* from a network interface. However, a minimal memory bandwidth will always be available for these devices, to prevent starvations. We can also assume in some cases computing cores can generate contention with each other, even without communications in parallel.

If we put together these hypotheses: when communications and computations executed in parallel reach together the memory bus bandwidth threshold, communication bandwidth starts to decrease to avoid impacting computing cores. When the assured minimum bandwidth for communications is reached, the performance of computations decreases uniformly between computing cores to fit the memory bus capacity; but the contention between the computing cores can already create contention penalizing computation performance too.

2.2 NUMA systems

Nowadays, machines commonly feature *Non-Uniform Memory Access* (NUMA): as depicted in Figure 1, multiple memory banks are available; each memory bank is connected to a single CPU. Although the whole available memory is still accessible through a single address space, the performance of a memory access varies whether a core is accessing its own memory or the memory from another memory bank. Hence, we will use the terms *local* and *remote* to qualify whether computing cores use memory respectively close or far to them.

The main consequence of such NUMA systems is that memory bandwidth will vary whether cores or network interface are accessing local or remote memory. Moreover, depending on where is located memory used for computations or communications, the path taken by the data between the NUMA node and the computing core or the network interface will be different, changing the locations of contention. Thus, our model has to take into account on which NUMA node data manipulated by computations or communications are located.

To focus on the interferences between computations and communications, we will not mix local and remote accesses from computing cores. This means we will model performance of computations and communications when cores of only one socket are computing. Considering computing cores of all sockets accessing the same NUMA node (thus, some of them are doing local accesses and other ones remote accesses) is another problematic that is left for future work.

2.3 Last level caches

The last-level cache (L3 cache on most machines), between cores and RAM memory, tends to alleviate the number of memory transfers done by computations. Thus, we would overestimate the number of memory movements if we assumed that every memory access instruction would lead to an actual transfer through the whole memory system. It would lead to inaccurate results about contention since our model only takes as input actual memory transfers.

If the data of the streams we are predicting the bandwidth go through the last-level cache, our model has to describe two phenomena: the contention on memory bus and the behaviour of the cache. However, the behaviour of the cache is complex to model [2, 6], implements undocumented strategies different for each processor manufacturer, and changes for each kind of application. All in all, modeling the cache is another topic, different from the one we are currently dealing with.

For all these reasons, we chose to ignore the last-level cache and make the data stream bypass it.

2.4 Modeling methods

A widespread model for contention is queuing theory [7, 15]: cores or network interfaces are customers; when they make a memory request, they enter in the queue: they leave the queue when the request is processed. Closed-form expressions exist for properties of such queues, especially the mean time spent in a queue. Unfortunately this kind of model is not relevant for our usecase. Since NUMA machines have a hierarchical organization of their memory, bottlenecks can appear on several places in the memory system (see Figure 1). Each place where contention can occur has to be represented by a dedicated queue, and the different queues of all memory components have to be combined to model the behaviour of the whole memory system. Correctly assembling the queues requires to have a sharp understanding of how the memory system works (knowledge usually not available publicly and specific to each processor generation and manufacturer). Even if we succeed in proposing an assembly of queues, getting all parameters of all queues would require lot of execution samples to be precise enough. Moreover, obtained parameters characterizing the queue can lack physical meaning, making the parameter interpretation harder. Most queuing models are built with the assumption that all customers have the same request rate; it is not necessarily true in our case: one network interface can issue memory requests at a higher rate than one computing core (a single computing core can reach a memory bandwidth of 5 GB/s, while network bandwidth can be around 10 GB/s). In such situation, we lose the closed-form expressions, which were the main advantage of queuing theory.

We chose a simpler model, easier to manipulate, but accurate enough for our needs: a basic threshold. While the bandwidth required by all issuers of memory requests stays under the memory bus capacity, there is no contention, no impact on performance. When it does not fit the memory bus anymore, only the bus capacity is available, and is split among computing cores and network interface. This model, described in detail below, has the advantages of requiring few application runs to calibrate its parameters, which are understandable with a physical meaning, well-known units, and coherent values regarding performed benchmarks and hardware features.

3 A model for memory bandwidth sharing

The model aims at giving the memory bandwidths available for computations and communications, and thus predicts the impact of the contention on their performance, using as input parameters the number of computing cores, the memory location of data used by computations and communications and the topology of the machine.

Since memory bandwidth depends on which NUMA node is accessed, we need to instantiate our model once for *local* accesses, noted $\mathcal{M}_{local}$ (*e.g.* memory for computations and communications located on the first NUMA node of the first socket), and once for *remote* accesses, noted $\mathcal{M}_{remote}$ (*e.g.* memory for computations and communications located on the first NUMA node of the second socket). Parameters of each model are defined with metrics collected on executions where data used for computations and communications are on the same NUMA node (case with the largest contention). Once parameters are collected, performance can be predicted according to these parameters. Performance with memory placement configurations other than the ones used to instantiate the model is predicted by combining the local and remote models. This section illustrates how the model is built, then summarizes model parameters and finally explains how models for local and remote accesses are combined to predict performance of all possible data placements.

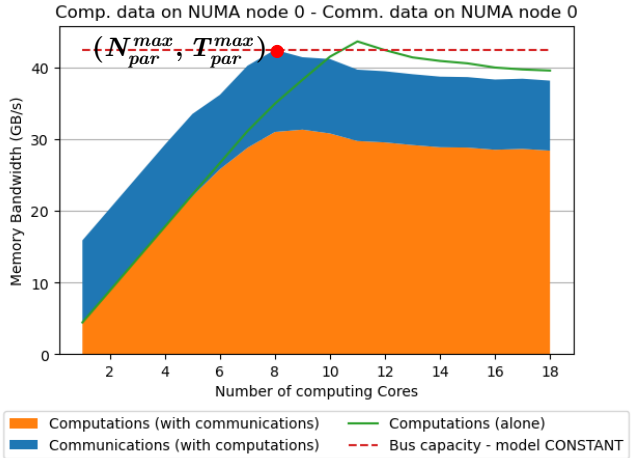


Figure 2: Stacked memory bandwidth for computations and communications with also memory bandwidth for only computations (green curve) and the memory bus capacity modeled as a constant (model Constant, red line).

3.1 Construction of the model

Memory bandwidths are predicted in two steps: first, $\mathcal{T}$ the total bandwidth the memory system can support according to the number of computing cores is predicted, then this total bandwidth is split between communications and computations.

A convenient way to understand how bandwidths which will be predicted by the model evolve is to sum memory bandwidths for computations and communications and visualize them by stacking them. Since both streams share parts of the same memory system, it allows to easily represent the share of the bus capacity among the two different streams. Figure 2 is an example of such representation: according to the number of computing cores, the orange area depicts memory bandwidth for computing cores and blue area depicts memory bandwidth for communications, when they are both executed in parallel. We also show the curve of the memory bandwidth for computation executed alone, in green.

3.1.1 Memory bus capacity

As a first approach, the maximum available memory bandwidth on the bus $\mathcal{T}$ can be set constant (version of the model called Constant) to the value corresponding to the maximum total memory bandwidth $\mathcal{T}_{\text{Constant}} = T_{par}^{max}$ when computations and communications are executed together. This maximum is reached with N_{par}^{max} computing cores (● on Figure 2). We can visualize this constant bus capacity with the red dashed line on Figure 2.

One can notice on Figure 2 that once memory contention begins, when there are too many cores accessing the memory, the capacity of the bus starts to decrease (*i.e.* different data streams are not able to completely fill the bus anymore). Since our model distributes the memory bus bandwidth between computations and communications, our estimation of the bus capacity has to be more accurate.

We can model this reduction of the memory bus capacity with a second version of the model, called Linear1: by considering that each additional computing core after N_{par}^{max} already computing cores reduces the memory bus capacity by δ. Thus, the maximum available bandwidth depends now on the number n of computing cores and can be expressed by the following equation:

$$\mathcal{T}_{\text{Linear1}}(n) = \begin{cases} T_{par}^{max} & \text{if } n \leq N_{par}^{max} \\ T_{par}^{max} - \delta \times (n - N_{par}^{max}) & \text{otherwise} \end{cases} \quad (1)$$

Such equation for $\mathcal{T}_{\text{Linear1}}(n)$ gives an approximation of the memory bus capacity as illustrated by Figure 3.

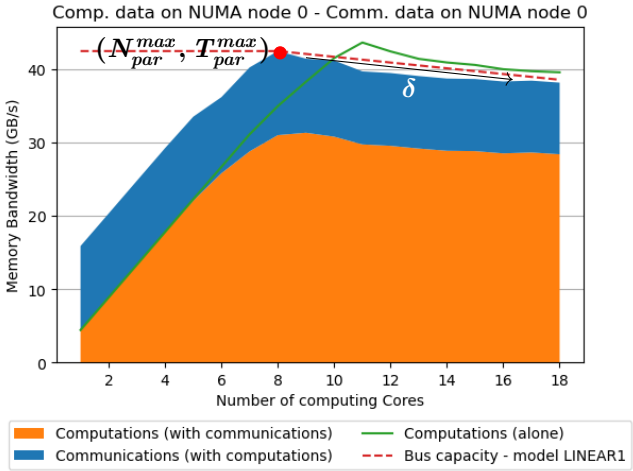


Figure 3: Stacked memory bandwidth for computations and communications with also memory bandwidth for only computations (green curve) and the memory bus capacity modeled as linearly decreasing after contention starts (model LINEAR1, red line).

Model version	Error of memory bus capacity prediction
CONSTANT	7.36 %
LINEAR1	2.32 %
LINEAR2	0.55 %

Table 1: Errors of different versions of the model to predict the memory bus capacity $\mathcal{T}$. Only points where contention occurs are considered.

Equation 1 for $\mathcal{T}_{\text{LINEAR1}}(n)$ does actually not catch the inflection point occuring after N_{par}^{max} computing cores and communications in parallel. This inflection point happens with the same number of computing cores than the number of computing cores N_{seq}^{max} necessary to reach the maximum memory bus bandwidth when there are only computations (● on Figure 4). With N_{seq}^{max} computing cores and communications in parallel, the total reached memory bandwidth is T_{par}^{max2} (● on Figure 4).

To reflect this inflexion point, we can have two different values for δ: δ_l and δ_r, depending whether there are respectively less or more than N_{par}^{max} computing cores. Hence, the model LINEAR1 becomes the model LINEAR2 to compute the memory bus capacity:

$$\mathcal{T}_{\text{LINEAR2}}(n) = \begin{cases} T_{par}^{max} & \text{if } n \leq N_{par}^{max} \\ T_{par}^{max} - \delta_l \times (n - N_{par}^{max}) & \text{else if } N_{par}^{max} < n \leq N_{seq}^{max} \\ T_{par}^{max2} - \delta_r \times (n - N_{seq}^{max}) & \text{otherwise} \end{cases} \quad (2)$$

$\mathcal{T}(n)$ given by the model LINEAR2 describes accurately the total memory bandwidth available, even in case of contention, as can be seen on Figure 4. This model is used in the remaining of the paper to compute the capacity of the memory bus.

Table 1 reports the prediction error of the different versions of the model to predict the memory bus capacity. The error is estimated with the mean absolute percentage error ($\frac{100\,\%}{n} \sum_{k=8}^{n} \left| \frac{a_k - p_k}{a_k} \right|$), comparing predictions (p_k) of the models and the measures (a_k). We only consider errors of points required later by the model, when contention occurs (starting from 8 computing cores in the illustrating example). Indeed, since we use as measure the sum of bandwidths for computations and for communications, which is very inferior to T_{par}^{max} as long as there is no contention, the error with less than 8 computing cores (in this example) would be very high and not really meaningful. Each refinement of the model reduces their prediction error.

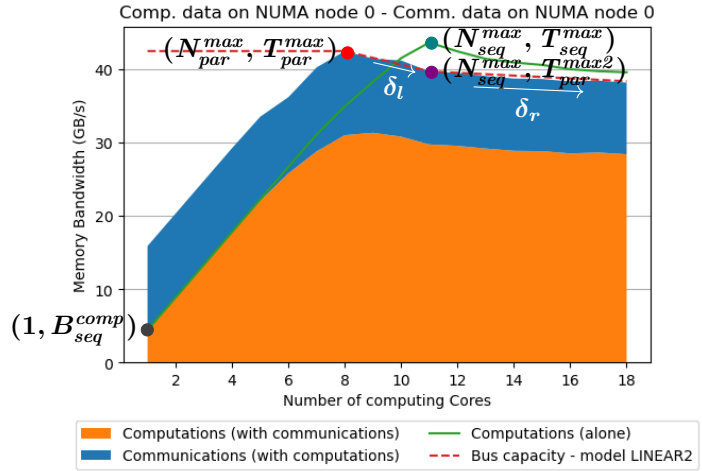


Figure 4: Stacked memory bandwidth for computations and communications with also memory bandwidth for only computations (green curve) and the memory bus capacity modeled as linearly decreasing after contention starts, taking into account the inflection point (model LINEAR2, red line).

3.1.2 Sharing memory bus bandwidth

Now we have an equation to describe the available memory bandwidth $\mathcal{T}_{\text{LINEAR2}}(n)$ according to the number of computing cores n, we need to share this bandwidth between computations and communications.

From the previous figures 2, 3 and 4, one can clearly observe computations do not get all the memory bandwidth they need when communications are executed in parallel (compare the orange area and the green curve). From this observation, the first approach is to give to communications all memory bandwidth they require (for any number n of computing cores: $\mathcal{B}_{par}^{comm}(n) = B_{seq}^{comm}$), and give the reminder to computations:

$$\mathcal{B}_{par}^{comp}(n) = \min(\mathcal{T}_{\text{LINEAR2}}(n) - \mathcal{B}_{par}^{comm}(n), n \times B_{seq}^{comp}) \tag{3}$$

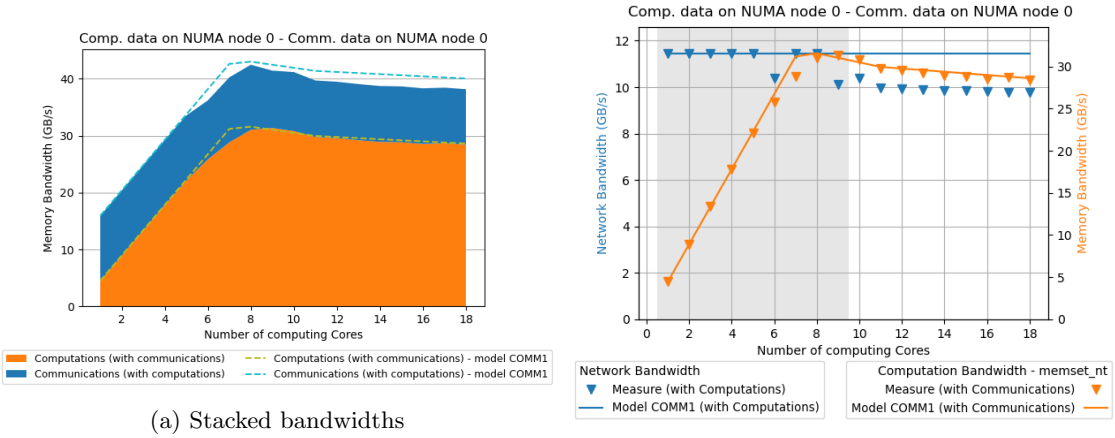
We add the constraint that computations cannot get more bandwidth than a perfect scaling (one computing core requires a bandwidth of B_{seq}^{comp}, ● on Figure 4), hence the min in the equation.

This first model of memory sharing is called COMM1, and is represented on Figure 5: Figure 5a uses the same visualisation of bandwidth sharing than previous figures; Figure 5b represents the same data, but in a different way to better see the accuracy of the model: according to the number of computing cores, measure (▼ markers) and prediction (continuous curve) of network bandwidth (in blue, to be read on the left Y-axis) and memory bandwidth for computations (in orange, to be read on the right Y-axis) are plotted. The latter representation allows to better compare with the measures the prediction of memory bandwidth for the two kinds of data streams.

On Figure 5b, one can notice that the prediction of network bandwidth is not correct when there is contention: the network bandwidth seems to decrease to a constant value, while the model COMM1 predicts always the same constant value, regardless the contention state.

Another version of the model, called COMM2, introduces a constant factor α reflecting the worse degradation of network bandwidth: $\alpha = \min_i(\frac{B_{par}^{comm}(i)}{B_{seq}^{comm}})$; $B_{par}^{comm}(i)$ is the network bandwidth measured when there are i computing cores in parallel.

However, the network bandwidth is reduced by α only when there is contention, not before. Depending on whether there are enough computing cores to generate contention or not, the memory bandwidth share is different. For any number of computing cores n, the bandwidth required to fit into the bus includes satisfying computing core requirements ($n \times B_{seq}^{comp}$) and assuring minimum



(a) Stacked bandwidths

(b) Split bandwidths

Figure 5: Bandwidth sharing for simultaneous computations and communications, with the model COMM1 giving all requested bandwidth to communications.

bandwidth to communications ($\alpha \times B_{seq}^{comm}$), it is noted $R(n)$ and is given by the following formula:

$$R(n) = n \times B_{seq}^{comp} + \alpha \times B_{seq}^{comm} \tag{4}$$

The share of the total bandwidth allocated to computing cores, for the model COMM2, is then described by the following equation:

$$\mathcal{B}_{par}^{comp}(n) = \begin{cases} n \times B_{seq}^{comp} & \text{if } R(n) < \mathcal{T}_{\text{LINEAR2}}(n) \\ \mathcal{T}_{\text{LINEAR2}}(n) - \mathcal{B}_{par}^{comm}(n) & \text{otherwise} \end{cases} \tag{5}$$

While all memory bandwidth requested by computing cores and minimal bandwidth assured for communications fit in the total available bandwidth, computations on n computing cores will get the memory bandwidth $\mathcal{B}_{par}^{comp}(n)$, corresponding to their request (perfect scaling). When the threshold $\mathcal{T}_{\text{LINEAR2}}(n)$ is reached, computations get the remaining bandwidth after communications got their share of the bandwidth.

The bandwidth for communications, for the model COMM2, is allocated as stated by the following equation:

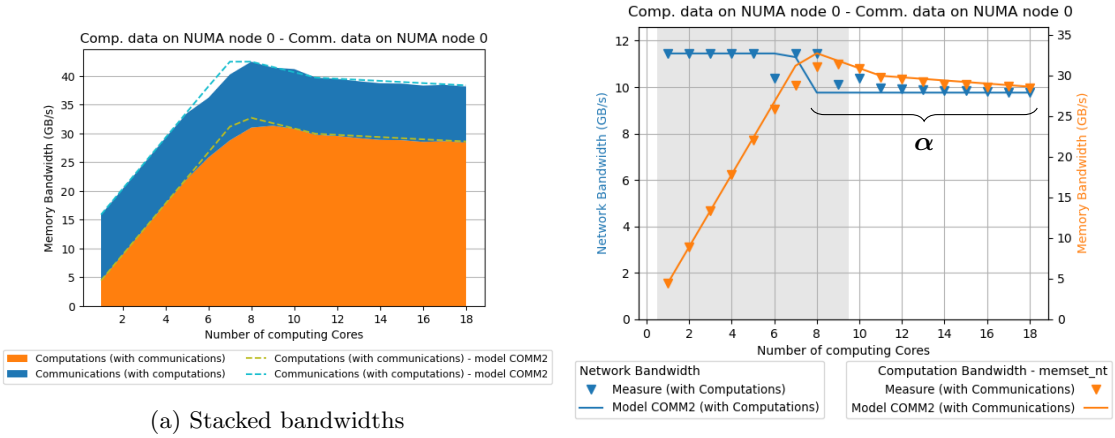
$$\mathcal{B}_{par}^{comm}(n) = \begin{cases} \min(\mathcal{T}_{\text{LINEAR2}}(n) - \mathcal{B}_{par}^{comp}(n), B_{seq}^{comm}) & \text{if } R(n) < \mathcal{T}_{\text{LINEAR2}}(n) \\ \alpha \times B_{seq}^{comm} & \text{otherwise} \end{cases} \tag{6}$$

While $R(n)$ is lower than the bus capacity $\mathcal{T}_{\text{LINEAR2}}(n)$, communications get the share of the total bandwidth unused by computing cores, but they cannot use more than the nominal performance of the network B_{seq}^{comm} (hence the min). When $R(n)$ exceeds the bus capacity, the bandwidth for communications is impacted by the factor α.

The model COMM2, using the equations 4, 5 and 6 and the model LINEAR2 to predict the memory bus capacity, is depicted on Figure 6. Compared to the model COMM1, the model COMM2 captures the constant degraded bandwidth given to communications when there is contention, yet the transition between the states without and with contention is sudden: in reality, the performance of the network does not drop so abruptly, by adding just one computing core.

To tackle this issue, one can make the α, used in equation 6, depend on the number of computing cores and linearly approximate its value between N_{par}^{max} and N_{seq}^{max}:

$$\alpha(n) = \begin{cases} \frac{\mathcal{B}_{par}^{comm}(i)}{B_{seq}^{comm}} - \frac{\frac{\mathcal{B}_{par}^{comm}(i)}{B_{seq}^{comm}} - \alpha}{N_{seq}^{max} - i} \times (n - i) & \text{if } N_{seq}^{max} - N_{par}^{max} > 1 \text{ and } n < N_{seq}^{max}, \\ & \text{where } i = \max_j(\{j | R(j) < \mathcal{T}_{\text{LINEAR2}}(j)\}) \\ \alpha & \text{otherwise} \end{cases} \tag{7}$$



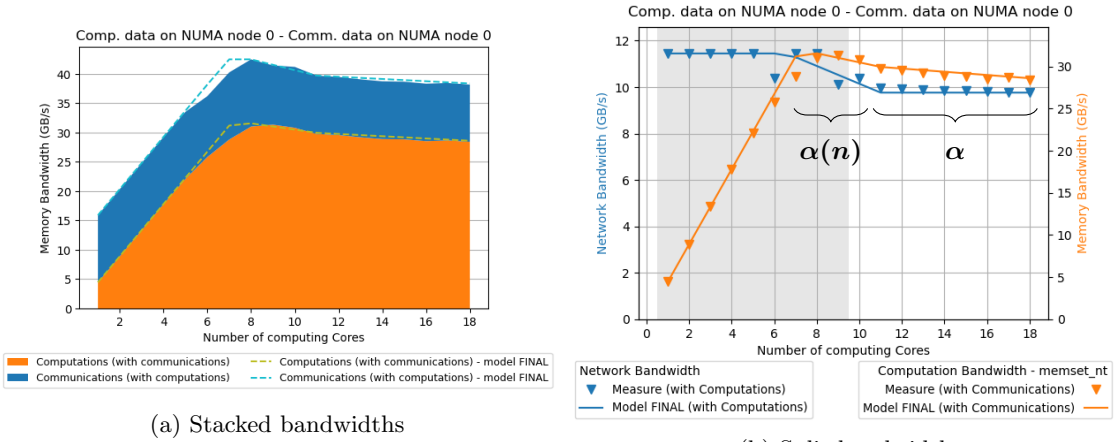
(a) Stacked bandwidths

(b) Split bandwidths

Figure 6: Bandwidth sharing for simultaneous computations and communications, with a model limiting bandwidth for communications. Communications are reduced by a constant factor α when there is contention.

With N_{seq}^{max} or more computing cores, communications get their minimal assured bandwidth, thus $\alpha(n) = \alpha$. When there are less computing cores, more than one core between N_{par}^{max} and N_{seq}^{max}, and $R(n) \geq \mathcal{T}_{\text{LINEAR2}}(n)$ (*i.e.* the case when $\alpha(n)$ has to be computed), bandwidth for communications does not abruptly drop to $\alpha \times B_{seq}^{comm}$. Therefore, we linearly interpolate the factor, with a line passing by the points where the factor of impact on communications with the maximum number of computing cores where $R(n) < \mathcal{T}_{\text{LINEAR2}}(n)$ is still valid (noted i in the equation), and α with N_{seq}^{max} computing cores.

Replacing $\alpha \times B_{seq}^{comm}$ in equation 6 by $\alpha(n) \times B_{seq}^{comm}$, with $\alpha(n)$ defined in equation 7, forms the model called FINAL, used in the remaining of this paper. Predictions with this model are depicted on Figure 7.



(a) Stacked bandwidths

(b) Split bandwidths

Figure 7: Bandwidth sharing for simultaneous computations and communications, with the model FINAL limiting bandwidth for communications. When there is contention, communications are slowly degraded by a factor $\alpha(n)$ until it reaches the constant value α.

Table 2 reports the prediction error of the different versions of the model to predict memory bandwidths for computations and communications. All model versions use the best model to get

Model version	Error of communication prediction	Error of computation prediction
COMM1	9.13 %	1.39 %
COMM2	2.37 %	1.57 %
FINAL	1.66 %	1.39 %

Table 2: Errors of the different versions of the model to predict the memory bandwidths for computations and communications (using the model LINEAR2 for $\mathcal{T}$).

the memory bus capacity: LINEAR2. The same formula than for table 1 is used, but with all predicted points (even when there is no contention). Again, each refinement of the model increases its precision.

To summarize the model predicting the bandwidth sharing between computing cores and simultaneous network communications: first, the memory bus bandwidth capacity is predicted, taking into account its reduction when contention occurs. Then, the memory bus capacity is shared between computing cores and network communications, respecting the following policy: while the sum of memory bandwidths requested by computations and communications is lower than the memory bus capacity, the requested memory bandwidth is granted; otherwise bandwidth for communications is reduced to a constant value and computing cores get the remaining memory bus capacity.

Moreover, to predict performance on all memory placement configurations, the model needs in some configurations to predict performance of computations and communications executed alone, when there is no contention. The bandwidth for communications executed alone is simply the model parameter B_{seq}^{comm}. The bandwidth for computations executed alone is given by the following formula:

$$\mathcal{B}_{seq}^{comp}(n) = \min(n \times B_{seq}^{comp}, \mathcal{T}_{\text{LINEAR2}}(n), T_{seq}^{max}) \tag{8}$$

The formula considers a perfect scaling of memory bandwidth allocated to computing cores, limited by the memory bus capacity $\mathcal{T}_{\text{LINEAR2}}(n)$ and cannot neither exceed the maximum bandwidth T_{seq}^{max} when computations are executed alone.

3.2 Summary of model parameters

The model requires several parameters, already mentionned previously, describing the behaviour of the machine when communications and computations are executed independently, to know nominal performance and predict them correctly when there is no contention, and in parallel, to know what can be the impact of contention. The following list summarizes them (most of them are annotated on Figure 4):

- $N_{par}^{max}, T_{par}^{max}$: the maximum total memory bandwidth T_{par}^{max} reached when computations and communications are executed simultaneously, and with how many computing cores N_{par}^{max} it is reached;
- $N_{seq}^{max}, T_{seq}^{max}$: the maximum memory bandwidth T_{seq}^{max} reached when computations are executed alone, and with how many computing cores N_{seq}^{max} it is reached;
- T_{par}^{max2}: the total memory bandwidth when communications are performed and N_{seq}^{max} cores are computing in parallel;
- δ_l, δ_r: the memory bandwidths lost per additional computing core when there are respectively between N_{par}^{max} and N_{seq}^{max} computing cores and when there are more than N_{seq}^{max} computing cores;
- B_{seq}^{comp}: the memory bandwidth used by one single computing core;
- B_{seq}^{comm}: the communication bandwidth when communications are executed alone;
- α: the ratio of the available bandwidth for communications in the worst case.

3.3 Model NUMA effect

NUMA systems present different memory bandwidths depending if accesses are made to a local or a remote NUMA node. Therefore, we need two model instantiations, each with its own set of parameter values. The set of parameters describing local accesses, when both computations and communications make memory accesses to the same local NUMA node (regarding to computing cores), is noted $\mathcal{M}_{local}$, and conversely, the set of parameters describing remote accesses, when they make memory accesses to the same NUMA node of a another socket, is noted $\mathcal{M}_{remote}$.

Using equations 2 to 7 (models Linear2 and Final), we can model performance for the two memory binding configurations we used to calibrate the two models (data for computations and communications both on the same NUMA node than the NIC and on the other NUMA node), by directly using the corresponding model. However, we need to combine these two models to predict performance on all other memory binding configurations. Predicting bandwidths of computations and communications requires now two additional parameters, to take into account data location: the index of the NUMA node where is located data used by computations (m_{comp}) and by communications (m_{comm}). These parameters, in addition to the number of NUMA nodes per socket noted $\#m$, allow to select the corresponding bandwidth according to placement.

In the rest of the section, the notation $\mathcal{B}(\mathcal{M})$ means the bandwidth $\mathcal{B}$ is given by using the model instantiation $\mathcal{M}$.

Regarding communications, the model to apply is selected with the following equation:

$$\mathcal{B}_{par}^{comm}(n, m_{comp}, m_{comm}) = \begin{cases} \mathcal{B}_{par}^{comm}(\mathcal{M}_{remote}, n) & \text{if } m_{comp} \geq \#m \text{ and } m_{comp} = m_{comm} \\ \mathcal{B}_{par}^{comm}(\mathcal{M}_{local} \setminus B_{seq}^{comm}(\mathcal{M}_{remote}), n) & \text{else if } m_{comm} \geq \#m \\ \mathcal{B}_{par}^{comm}(\mathcal{M}_{local}, n) & \text{otherwise} \end{cases} \tag{9}$$

If both computations and communications access to the same remote NUMA node, communication bandwidth is given by the remote model $\mathcal{M}_{remote}$. In all other cases, communications are less subject to contention and follow the local model $\mathcal{M}_{local}$. However, on some machines, the network performance is very sensible to the locality of exchanged data. Since $\mathcal{M}_{local}$ is instantiated with communication bandwidth with data located on the local NUMA node, it may not fit the network performance when data for communications are located on the remote NUMA node. Therefore, in this case, we use the local model, but with the nominal network performance when data is located on remote memory, *i.e.* the B_{seq}^{comm} of $\mathcal{M}_{remote}$.

The model for computation bandwidth is selected with the following equation:

$$\mathcal{B}_{par}^{comp}(n, m_{comp}, m_{comm}) = \begin{cases} \mathcal{B}_{par}^{comp}(\mathcal{M}_{local}, n) & \text{if } m_{comp} < \#m \text{ and } m_{comp} = m_{comm} \\ \mathcal{B}_{seq}^{comp}(\mathcal{M}_{local}, n) & \text{if } m_{comp} < \#m \text{ and } m_{comp} \neq m_{comm} \\ \mathcal{B}_{par}^{comp}(\mathcal{M}_{remote}, n) & \text{if } m_{comp} \geq \#m \text{ and } m_{comp} = m_{comm} \\ \mathcal{B}_{seq}^{comp}(\mathcal{M}_{remote}, n) & \text{if } m_{comp} \geq \#m \text{ and } m_{comp} \neq m_{comm} \end{cases} \tag{10}$$

Computations are impacted by contention only when data used for communications are on the same NUMA node as data for computations. In such case, bandwidth for computations is the one with communications in parallel $\mathcal{B}_{par}^{comp}$, from the model corresponding to computation data location, local or remote. In the same fashion, when computations and communications do not use the same NUMA node for their data, computations get their nominal memory bandwidth $\mathcal{B}_{seq}^{comp}$.

Appendix A presents algorithms to predict memory bandwidth for computations and communications, implemented using equations described above.

4 Evaluation of the model

We want to measure the impact of memory contention on computations and communications, when they are executed in parallel, and to compare it with the predictions of our model. To know

the impact, we need the performance of computations and communications executed alone and in parallel.

4.1 Experimental setup

4.1.1 Benchmarking program

We designed our own benchmarking suite[1], which executes the following steps for all possible number of computing cores:

1. Computations alone
2. Communications alone
3. Computations and communications in parallel

Computations are spread among cores dedicated to computations with OpenMP pragmas and communications are done by a single thread bound to a dedicated core, between two machines using MPI. We used MadMPI, the MPI interface of NewMadeleine [3], for the presented results, but similar results are observed with other MPI libraries, such as OpenMPI. Computations and communications use different data, making them completely independent.

Having several computing cores and one core dedicated to communication management mimics the working of runtime systems such as StarPU [1] or PaRSEC [4]. It has been demonstrated that using threaded communication [11, 12] allows communications and computation overlap and thus better application performance.

Performance is measured on a single node, but we still need two machines for network exchanges. We study the performance penalty caused by memory contention, therefore, to control and understand memory movements, all computing cores perform *non-temporal* `memset` instructions to move data from cores to memory, and communication performance is measured with the bandwidth observed to receive messages of 64 MB from the other machine. We use *non-temporal* instructions to bypass the last level cache, as explained in section 2.3: they tell the processor to store data directly in memory, bypassing the cache. Data used for communications and computations are explicitly bound on selected NUMA nodes, to know the data location and consider it in the model. Memory bandwidth for computations is computed from the duration of the `memset` instructions, each computing core always work on the same amount of data (weak scaling). Memory bandwidth for communications is considered to be the same as the network bandwidth, *i.e.* the message size over the necessary time to receive data from the other machine, since this stream has also to go through the memory bus after arriving on the network interface. Only samples collected during steady state are considered: all cores execute computation iterations for a defined amount of time, then we skip performance of first and last iterations of each core, to get rid of the performance when not exactly all cores are computing.

To bind memory on a specific NUMA node, bind threads to cores and gather topology information, we use Hwloc [5].

4.1.2 Modeling all placements

With NUMA machines, we need two model instantiations: one for local memory accesses and another one for remote accesses. On a machine with two sockets (processors) and two NUMA nodes per socket, we would execute our benchmarking program to get model parameters using memory for computations and communications both located on the first NUMA node of the first socket for the local model and using memory located on the first NUMA node of the second socket for the remote model. Thus, we need to measure memory bandwidths of two placement configurations, to latter be able to predict performance of all other configurations (16 in this example, since there are 4 possibilities where to put data for computations and the same 4 possibilities for communication data), as explained in section 3.3.

[1] Available on `https://gitlab.inria.fr/pswartva/memory-contention`

Platform Type	Name	Processor	Memory	Network
Experimental	billy	2 × AMD EPYC 7502 with 32 cores	128 GB of RAM 2 NUMA nodes	InifiniBand ConnectX-6 HDR
	henri	2 × Intel Xeon Gold 6140 with 18 cores	96 GB of RAM 2 NUMA nodes	InifiniBand ConnectX-4 EDR
	henri-subnuma		96 GB of RAM 4 NUMA nodes	
Medium-scale	bora	2 × Intel Xeon Gold 6240 with 18 cores	192 GB of RAM 2 NUMA nodes	Omni-Path HFI Silicon 100 series
	dahu	2 × Intel Xeon Gold 6130 with 16 cores	192 GB of RAM 2 NUMA nodes	Omni-Path HFI Silicon 100 Series
	diablo	2 × AMD EPYC 7452 with 32 cores	256 GB of RAM 2 NUMA nodes	InifiniBand ConnectX-6 HDR
	grvingt	2 × Intel Xeon Gold 6130 with 16 cores	192 GB of RAM 2 NUMA nodes	Omni-Path HFI Silicon 100 Series
	pyxis	2 × Cavium-ARM ThunderX2 99xx with 32 cores	256 GB of RAM 2 NUMA nodes	InifiniBand ConnectX-6 EDR
Production	occigen	2 × Intel Xeon E5 2690v4 with 14 cores	64 GB of RAM 2 NUMA nodes	InifiniBand Connect-IB FDR

Table 3: Characteristics of testbed platforms.

The program to measure memory bandwidth of one placement configuration needs to be executed for all possible numbers of computing cores, in the range of the number of cores on the first socket, as explained in section 3. Once the performance metrics (memory bandwidth for computations alone and in parallel of communications, network bandwidth for communications alone and in parallel of computations) are extracted from benchmark outputs, the evolution of the bandwidths over the number of computing cores is analyzed (it mostly looks for minimums and maximums) and the parameters of the model, listed in section 3.2, are computed.

Note that this process can be optimized: once the peaks of bandwidth T_{par}^{max} and T_{seq}^{max} are found, one can skip executions with number of computing cores greater than N_{seq}^{max}, except the execution with all cores of the first socket, required to compute δ_r. Here we still need to execute the program with all possible numbers of computing cores, to evaluate the accuracy of our model.

4.1.3 Testbed platforms

We evaluated our model for the bandwidth metrics obtained on several platforms with different characteristics: from small experimental platforms to large production ones. Since we target HPC systems, we considered only fast networks, where contention occurs more; with technologies such as InifiniBand and Omni-Path. Table 3 describes characteristics of platforms used to measure model accuracy. `henri` and `henri-subnuma` are the same platform, allowing to access to the BIOS to change number of NUMA nodes. Hyperthreading is only enabled on platforms `dahu`, `grvingt`, `pyxis` and `occigen`, however, on all platforms, threads are bound to physical cores (*i.e.* hyperthreads are not used).

Values of model parameters for each platform are presented in Appendix B.

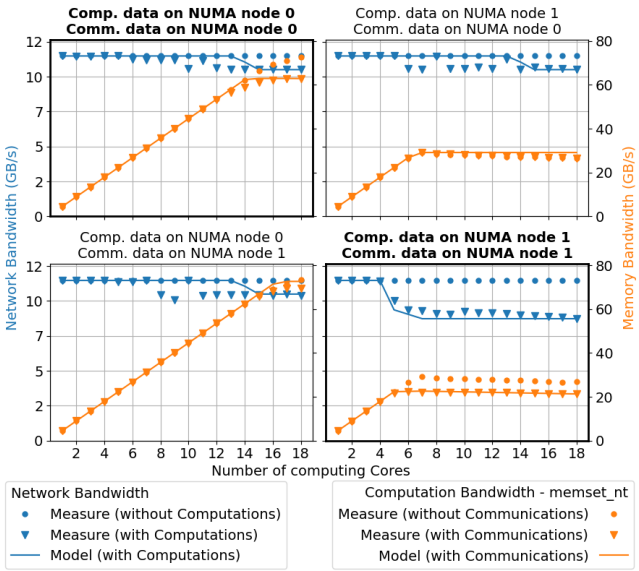


Figure 8: Performance of computations and communications along with our model prediction on `henri` (Intel, InifiniBand).

4.2 Results

Figures 8 to 13 depict performance of computations and communications as well as the model predictions. Each figure is composed of several subplots, one per possible placement combination of data for computations and data for communications on available NUMA nodes. For instance, Figure 8 represents results on the `henri` platform, with 2 NUMA nodes. Data for communications can be located on 2 NUMA nodes, as well as data for computations, which leads to 4 placement combinations. Each line of plots represents one placement for communication data, while columns represent placements for computation data. Titles above each plot precise the placement of data. The two placement combinations used to instantiate the local and remote models are highlighted with a bold title and a thicker frame. Each subplot presents, according to the number of computing cores, network bandwidth (in blue, to be read on the left Y-axis) and memory bandwidth for computations (in orange, to be read on the right Y-axis), when they are executed alone (• markers) and in parallel (▼ markers). The blue and orange curves indicate our model predictions of the bandwidth for respectively communications and computations. Error bars are not shown to ease reading, but the run-to-run variability is very low.

`henri` Figure 8 shows there is contention between computations and communications, impacting them both, more or less severely according to data placement. Our model is accurate when computations and communications perform both remote memory accesses. When computations and communications perform both local accesses, our model reflects the correct impact on communications too late (the model predicts a decrease starting with 14 computing cores, while it is 10 in reality), because communications start to be impacted before the total bandwidth threshold $\mathcal{T}$ is reached. Other memory placement configurations, not used to instantiate the model, show the same flaws.

`henri-subnuma` The `henri` platform configured with 4 NUMA nodes allows 16 data placement combinations, described by Figure 9. The grey and white areas are used to distinguish the two NUMA nodes of the first socket. With such numerous configurations, symmetries in performance appear, mimicking symmetries of the machine topology: for instance, when data for computations and communications are on different NUMA nodes of the second socket (right half of set of plots), performance is always the same, regardless of which NUMA nodes are used. These symmetries allow

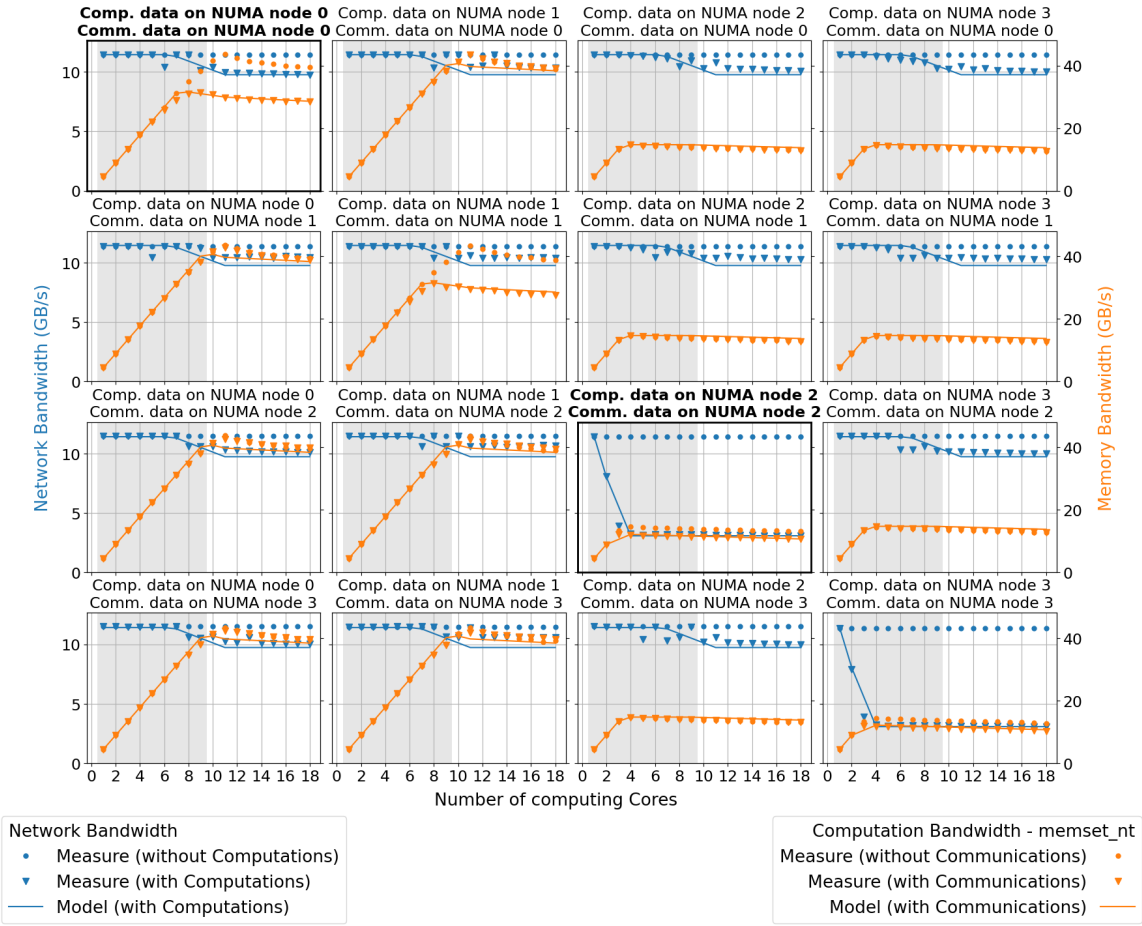


Figure 9: Performance of computations and communications along with our model prediction on `henri-subnuma` (Intel, InfiniBand).

the predictions done by the model to be correct, with only two configurations used to predict all 16 combinations. All these configurations also show that the placement configurations the most disturbed by memory contention are the ones where data for computations and communications are on the same NUMA node (*i.e.* subplots on the diagonal of the figure), while computations are almost not impacted in other cases. Therefore we can guess memory contention occurs the most on memory controllers (responsible of accesses to one dedicated NUMA node), rather that on the inter-socket connection bus.

Figures used to explain the model in previous section correspond to the top left subplot of the Figure 9.

`diablo` Figure 10 shows results on the `diablo` platform, and illustrates especially the case when network performance are highly sensible to data locality: when data for communications is on the first NUMA node, network bandwidth reaches only 12.1 GB/s whereas when data is on the second NUMA node (which the NIC is actually plugged to), network bandwidth can raise up to 22.4 GB/s. Our model succeeds in predicting performance, even if there is almost no contention on this platform.

`billy` Figure 11 depicts results on the `billy` platform, similar to `diablo`. The network performance is still sensible to placement (we get stable 14 GB/s when data for communications is on the first NUMA, up to 20 GB/s otherwise), but is more chaotic (when data for communications is on the second NUMA node, network bandwidth oscillates between 12 GB/s and 24 GB/s); the model

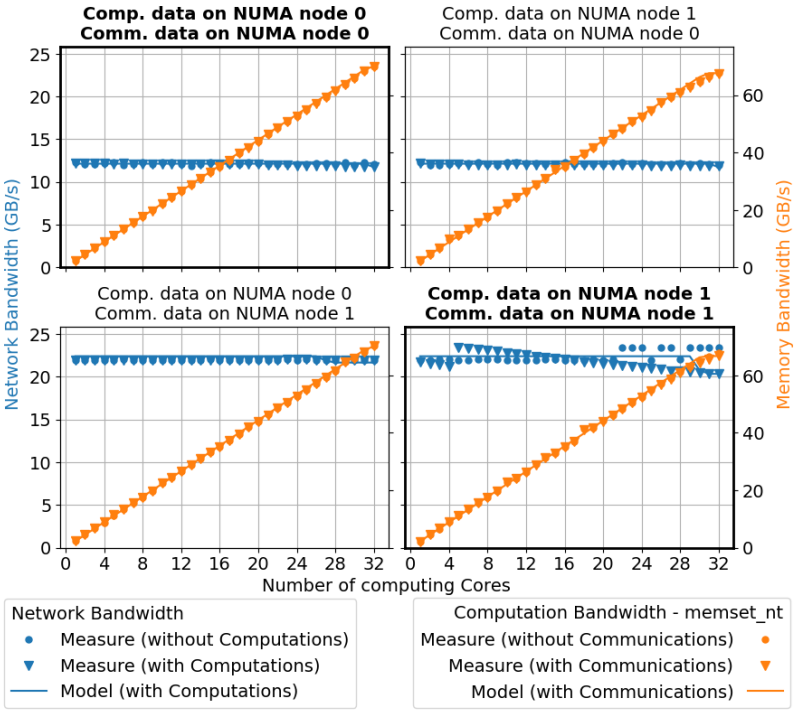


Figure 10: Performance of computations and communications along with our model prediction on `diablo` (AMD, InifiniBand).

fails to capture the variability, but follows the general trend of observations.

`occigen` Figure 12 shows results on the only production platform of our testbed. On this ancient platform (2014-2022), only computations are impacted when computations and communications do both remote memory accesses. For instance, with 7 computing cores, memory bandwidth for computations decreases from 21.1 GB/s to 18.5 GB/s when communications are executed in parallel. Network bandwidth stays always constant at 6.2 GB/s. This platform is where our model is the most accurate, with the lowest prediction error (see further).

`pyxis` Figure 13 shows results on a platform with ARM processors. Our model predicts correctly performance of computations, although it does not catch that memory bandwidth for computations does not scale well when it gets closer to the threshold. For instance, with data for computations on the first NUMA node and data for communications on the second node (bottom left plot), our model predicts a memory bandwidth of 62.5 GB/s for 19 computing cores, while in reality is 58.7 GB/s. Network performance is not correctly predicted for placement configurations which were not used to instantiate the model. On this architecture, the network performance seem to be harder to predict by just relying on the locality of the data.

`bora, dahu, grvingt` Model predictions for platforms equipped with similar hardware (Intel processor and Omni-Path network) give as expected similar results, as can be seen on figures 14, 15 and 16.

Table 4 reports the prediction error on all platforms. The error is estimated with the mean absolute percentage error ($\frac{100\,\%}{n}\sum_{k=1}^{n}\left|\frac{a_k-p_k}{a_k}\right|$), for predictions of computations and communications separately, by distinguishing also predictions made by the model on a placement configuration used to instantiate the model (*samples*) or not (*non-samples*). Regarding communications, the highest prediction error on all configurations is on `billy` (8.22 % on sample configurations, 10.84 % on

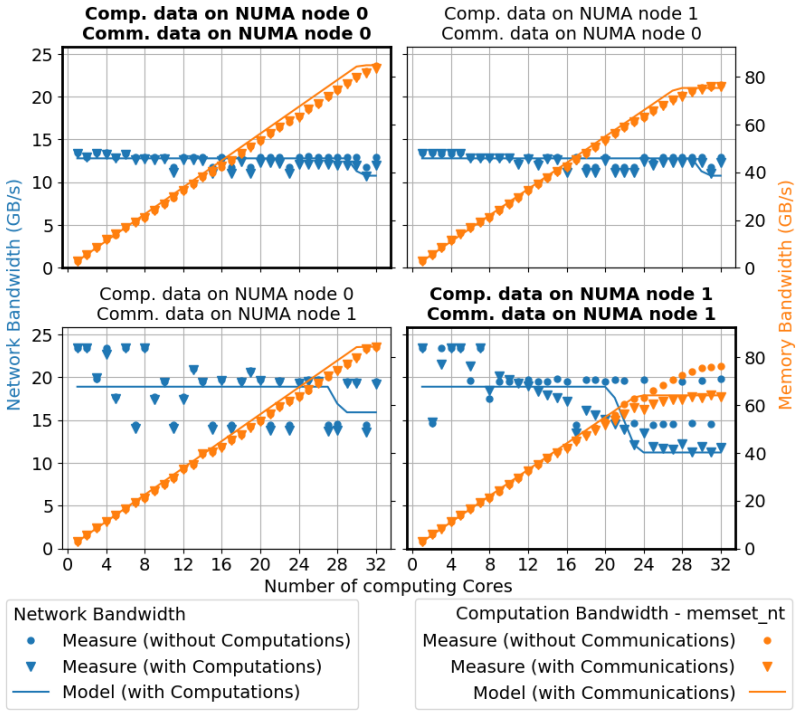


Figure 11: Performance of computations and communications along with our model prediction on `billy` (AMD, InifiniBand).

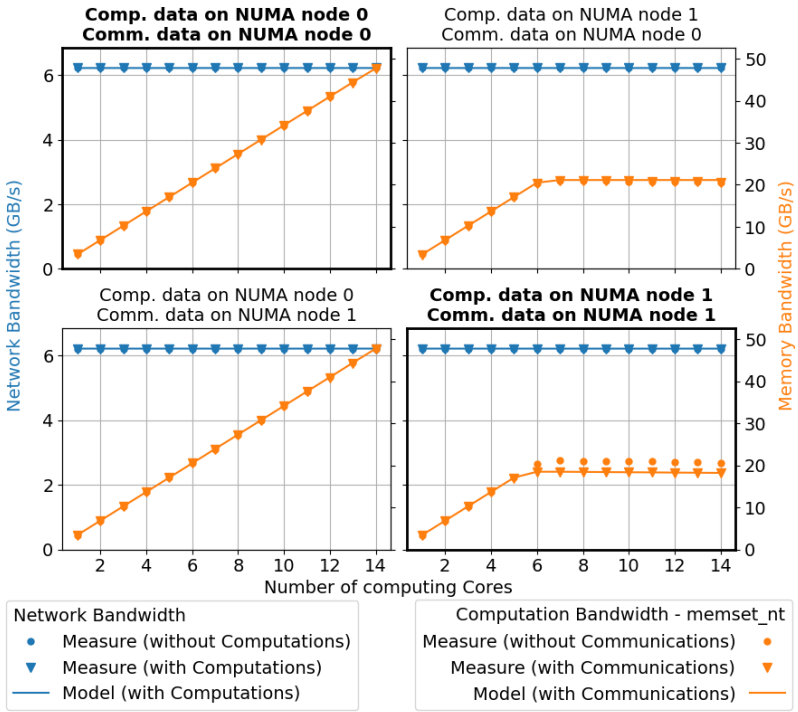


Figure 12: Performance of computations and communications along with our model prediction on `occigen` (Intel, InifiniBand).

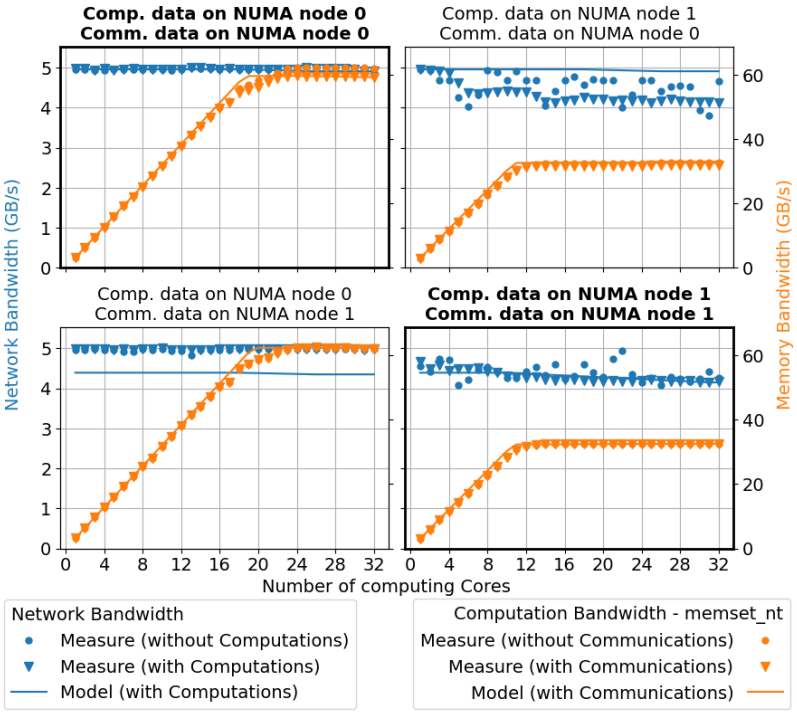


Figure 13: Performance of computations and communications along with our model prediction on `pyxis` (ARM, InifiniBand).

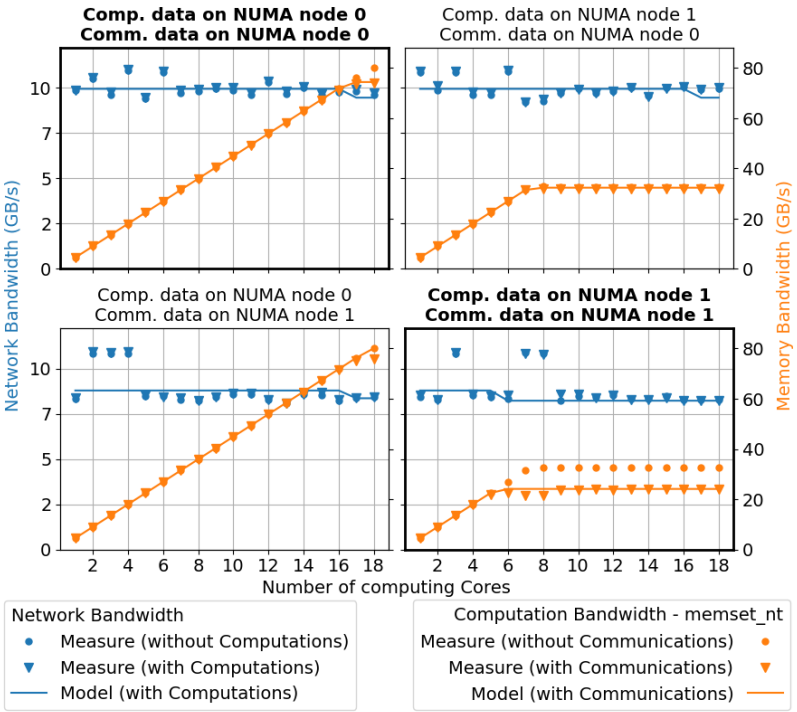


Figure 14: Performance of computations and communications along with our model prediction on `bora` (Intel, Omni-Path).

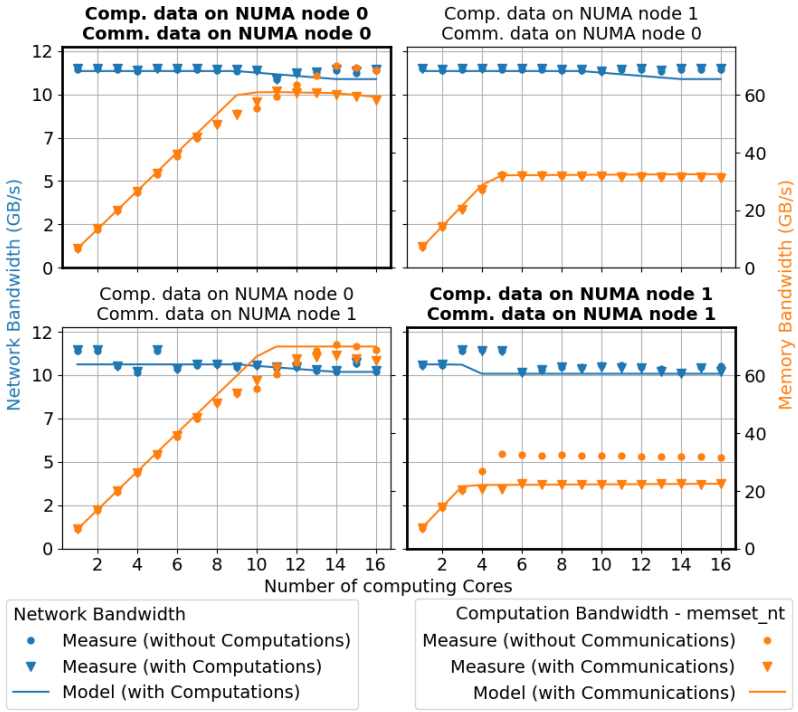


Figure 15: Performance of computations and communications along with our model prediction on `dahu` (Intel, Omni-Path).

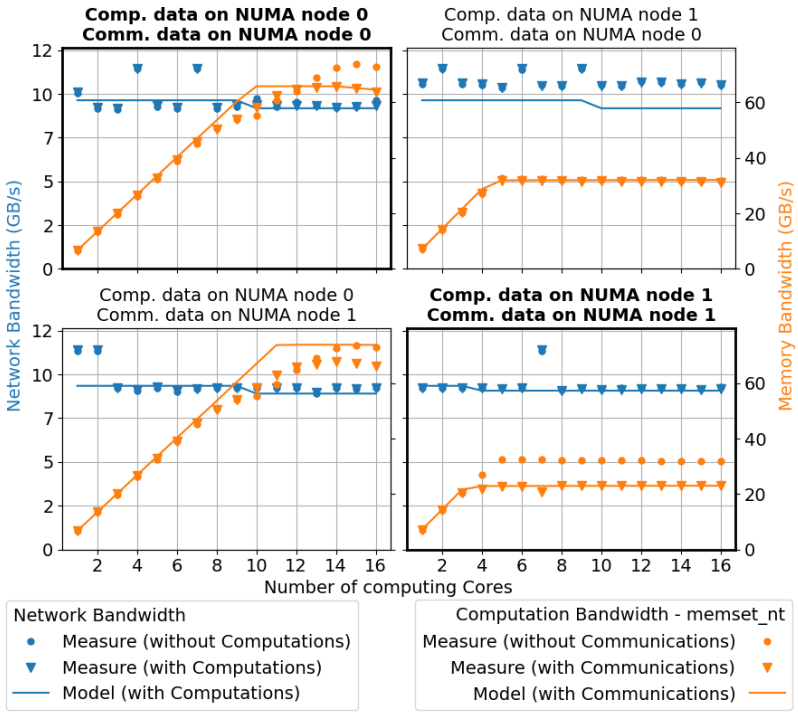


Figure 16: Performance of computations and communications along with our model prediction on `grvingt` (Intel, Omni-Path).

Platform	Communications			Computations			Average
	on Samples	*on non-Samples*	*all*	*on Samples*	*on non-Samples*	*all*	
`henri`	2.62 %	3.53 %	3.08 %	0.80 %	2.34 %	1.57 %	2.32 %
`henri-subnuma`	2.90 %	3.80 %	3.69 %	1.89 %	3.66 %	3.44 %	3.56 %
`billy`	8.22 %	10.84 %	9.53 %	3.98 %	3.40 %	3.69 %	6.61 %
`bora`	4.39 %	5.14 %	4.77 %	1.34 %	0.78 %	1.06 %	2.91 %
`dahu`	2.76 %	2.38 %	2.57 %	2.00 %	3.85 %	2.92 %	2.74 %
`diablo`	2.32 %	1.54 %	1.93 %	0.92 %	0.99 %	0.95 %	1.44 %
`grvingt`	3.41 %	8.06 %	5.74 %	2.44 %	4.48 %	3.46 %	4.60 %
`pyxis`	1.15 %	13.32 %	7.24 %	1.95 %	2.79 %	2.37 %	4.80 %
`occigen`	0.01 %	0.01 %	0.01 %	0.22 %	0.58 %	0.40 %	0.20 %
Average	3.09 %	5.40 %	4.28 %	1.73 %	2.54 %	2.21 %	**3.24 %**

Table 4: Model errors on testbed platforms.

others), explainable by the high variability of network performance, even when communications are executed alone, not taken into account by our model. The prediction error is also high on `pyxis`, especially on non-sample configurations (13.32 %), caused by the wrong appreciation of locality impact on this architecture, as discussed above. On other platforms, the average of prediction error of network bandwidth on all placement configurations is below 6 %. Performance of computations is better predicted, with an overall error lower than 4 %. Worst cases are on `billy` (3.69 %) and `grvingt` (3.46 %), where the model tends to over-estimate the bandwidth for computations because it assumes a perfect scaling when the number of computing cores increases, but in reality computing cores start to contention before reaching the bandwidth threshold.

4.3 Discussion

Results presented above show our model is valid to predict memory bandwidth allocated to communications and to computations: from sample executions on two different placement configurations to instantiate the whole model, we are able to predict bandwidths with all possible placement configurations, with an overall prediction error lower than 4 %. Higher prediction errors come most often from unstable input data, nonetheless the model formulation allows us to better understand in which circumstances memory contention happens and how the hardware deals with it.

4.3.1 Model limits

Even though our model makes overall good predictions, there are some corner cases where it show its limits. It has difficulties to accurately predict network bandwidth if network performance is not stable even without contention (see for instance results on `billy`, `pyxis`, `bora` or `grvingt`). On systems where data locality can highly influence network performance (such as `diablo`, `billy`, `pyxis` or `bora`), the model can be wrong more often, especially on placement configurations not used to instantiate the model. These weaknesses are not related to modeling of contention, since the odd network performance is also hard to predict with communications executed alone. Being able to model network performance in all placement configurations, when communications are executed alone, would help improving our model, to predict network bandwidth in case of contention.

On machines with many NUMA nodes (more than 4; for instance, `billy` can be configured with 8 NUMA nodes – 4 per processor), network performance under memory contention depends on data locality and the heuristic given by formula 9 is not sufficiently accurate anymore. Moreover, when communications and computations use the same NUMA node for their data (*i.e.* when contention has the most impact), the distribution of memory bandwidth between computations and communications before the threshold is reached (first cases of equations 5 and 6) is, in our model, more in favour of computations as in reality. Thus, these more complex system topologies would require more precise

hypotheses about memory routing between NUMA nodes to model them accurately. Moreover, evaluating the model on all placement configurations (64 with 8 NUMA nodes!) is more difficult due to necessary time to execute benchmarks on all configurations (about one hour per configuration).

The model predictions are only valid for the parameters of the benchmarks used to instantiate the model: the computation kernels executed by computing cores and the message size used by communications. For different computation kernels and message sizes, memory contention can be different and thus model parameters as well. However, since the computation kernels and message size were chosen here to maximize the contention, other kernels or message size should produce less contention, but the insights provided by our model in the worst case should still be valid.

4.3.2 Lessons learned

Distinguishing location of data used for computations and for communications allows to change paths of the two different data streams in the memory system and thus better locate bottlenecks, where memory contention occurs. First hypotheses assume contention happens in memory controller (controlling the memory of a NUMA node) or in inter-processor link. Results on machines with 2 NUMA nodes show contention occurs when data for communications and computations are located on the same NUMA node, especially on the same *remote* NUMA node (*i.e.* data streams have to go through inter-processor link and memory controller). When communications and computations use each their own NUMA node for their data, memory contention is very low (when not null). Results on machines with 4 NUMA nodes (2 *local* and 2 *remote* nodes, for instance on `henri-subnuma`), refine the location of the bottleneck: when computations and communications do both remote accesses (data streams have to go through the inter-socket link), performance is the most impacted due to contention when they use the same remote NUMA node. Thus, **the place where the most contention occurs is memory controller, and not the inter-socket link**.

The hypotheses made to design the model, and validated with experiments, teach us **memory bandwidth for network communications is the first reduced** in case of memory contention, to preserve memory bandwidth dedicated to computations. However, a **minimum bandwidth is always assured for network**, to prevent starvations. When this minimum bandwidth is reached, bandwidth for computations starts to decrease to fit memory system capacity.

5 Related work

Our previous work [8] focused on the different factors impacting the memory contention between computations and communications: data and thread placement, message size and arithmetic intensity of computing kernels. At that time, we only reported the results of our observations, without any attempt to model the phenomenon.

Since literature about contention between computations and communications is pretty rare, works about its model is even more sparse.

A theoretical model of the memory bandwidth sharing between computing and communicating threads was made by Langguth *et al.* [13]. Although they considered communications and computations are executed simultaneously, in their model, when communications end before computation, computation gets again all the available bandwidth and *vice-versa* when computation ends before communications. We rather focus on the steady state when there are always computations and communications in parallel, by considering bandwidths instead of durations. Moreover our model is more low-level, by considering the data placement on the machine topology and the number of computing cores.

Works presented in the rest of this section did not consider communications, but were helpful to better understand the memory system, and the possibilities to model its behaviour, especially under contention.

Queuing theory is often used [7, 15] to model memory contention. Each queue can represent one contention point, and assembling them can describe the general behaviour of the whole memory system. Model parameters are derived from hardware counters, read while executing applications.

This kind of model fits well with homogeneous queue consumers (computing cores, caches, memory controllers), but is more difficult to use in our context, as explained in section 2.4.

Wang *et al.* presented [16] the possible bottlenecks in the memory system to model them with Integer Programming, to find the optimal number of cores to execute memory-bound applications, especially on NUMA systems.

Majo and Gross studied [14] the behaviour of memory controllers in charge of serving local and remote memory accesses. They distinguished the local memory bandwidth (of the local memory controller) and the remote memory bandwidth (of the QPI bus) and modeled the maximum available bandwidth as a pondered sum of the two bandwidths, by introducing a *sharing-factor*. The evolution of this factor depending on the number of computing cores helps to understand how the memory controller manages its queuing fairness between different types of memory requests.

Goodman *et al.* presented [10] Pandia, a framework to predict performance of other configurations (number of threads and their placement) of parallel applications. From a machine description and 6 well-chosen application runs, they have all required information to make accurate predictions, by knowing the bandwidth capacity of the different memory buses. They take into account parallel fraction, memory accesses, load balancing and computing resource demands of applications, and rely on hardware counters to get these information.

All in all, our work sets oneself apart by modeling memory bandwidths available for computations and communications, when they are executed simultaneously. We expand our model to predict memory bandwidths according to location of memory used for computations or communications. We instantiate our model without consulting hardware counters, by executing only two benchmarks.

6 Conclusion

Computations and communications in parallel distributed HPC applications can be executed in parallel to save execution time. With memory-bound computations and network exchanges with large messages, contention can occur in the memory system, reducing performance of both computations and communications.

In this paper, we proposed a model to predict memory bandwidth allocated for computations and communications when they are executed in parallel. Predictions are made from parameters describing behaviour of the memory system with two data placement configurations. From these parameters, the topology description of the machine and information about data locality, our model is able to predict memory bandwidth for computations and for communications, regardless on which NUMA node data are located, with an average prediction error lower than 4 %.

Building this model allows to better understand that memory contention is the most severe when computations and communications use data located on the same NUMA, bottleneck causing this contention is mainly located in the NUMA node controller, rather than in the inter-socket connection bus. In case of contention, the system first degrades memory bandwidth allocated to communications, but ensures a minimum, and then reduces computation bandwidth if necessary.

As future work, we would like to improve our model to better deal with the impact of data locality and study the consequences on the model if we relax its constraints: for instance if application performs communications with bidirectional data movements (*i.e. ping-pongs* instead of only *pongs*), as well as similar computing kernels (*e.g.* copying an array into another instead of just initializing an array with a single value). We also would like to take into account the last level cache into our model. Designing this model was a first step to consider memory contention between computations and communications in distributed HPC applications. Taking into account this behaviour directly into application can be challenging, the solution is rather to exploit indications provided by the model in runtime systems: they could better know on which NUMA node store data and how many computing cores should be used to avoid memory contention.

Software Availability We endeavor to make our experiments reproducible. A public companion[2] contains the instructions to reproduce our study.

Author contributions

Alexandre Denis: Conceptualization, Methodology, Resources, Writing - Original Draft.
Emmanuel Jeannot: Conceptualization, Supervision, Writing - Original Draft
Philippe Swartvagher: Software, Data Curation, Investigation, Methodology, Visualization, Writing - Original Draft.

Acknowledgment

This work is supported by the Agence Nationale de la Recherche, under grant ANR-19-CE46-0009.

This work is supported by the Région Nouvelle-Aquitaine, under grant 2018-1R50119 *HPC scalable ecosystem*.

Experiments presented in this paper were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see `https://www.plafrim.fr/`).

This work was granted access to the HPC resources of CINES under the allocation 2021-A0100601567 attributed by GENCI (Grand Equipement National de Calcul Intensif).

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`).

The authors furthermore thank Brice Goglin and Guillaume Pallez for their help and advice regarding this work.

References

[1] Emmanuel Agullo, Olivier Aumage, Mathieu Faverge, Nathalie Furmento, Florent Pruvost, Marc Sergent, and Samuel Thibault. Achieving High Performance on Supercomputers with a Sequential Task-based Programming Model. *IEEE Transactions on Parallel and Distributed Systems*, 2017.

[2] Diego Andrade, Basilio B. Fraguela, and Ramón Doallo. Accurate prediction of the behavior of multithreaded applications in shared caches. *Parallel Comput.*, 39(1):36–57, jan 2013.

[3] Olivier Aumage, Elisabeth Brunet, Nathalie Furmento, and Raymond Namyst. NewMadeleine: a Fast Communication Scheduling Engine for High Performance Networks. In *Workshop on Communication Architecture for Clusters (CAC 2007)*, Long Beach, California, United States, March 2007.

[4] George Bosilca, Aurelien Bouteiller, Anthony Danalis, Thomas Herault, Pierre Lemarinier, and Jack Dongarra. Dague: A generic distributed dag engine for high performance computing. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 1151–1158, 2011.

[5] François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst. hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications. In IEEE, editor, *PDP 2010 - The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing*, Pisa, Italy, February 2010.

[2] `https://gitlab.inria.fr/pswartva/paper-model-memory-contention-r13y`, archived on `https://www.softwareheritage.org/` with the ID `swh:1:snp:109abdd69ee0fc7d2c4d891fda574261ddd7df5d`

[6] Dhruba Chandra, Fei Guo, Seongbeom Kim, and Yan Solihin. Predicting inter-thread cache contention on a chip multi-processor architecture. In *11th International Symposium on High-Performance Computer Architecture*, pages 340–351, 2005.

[7] Younghyun Cho, Surim Oh, and Bernhard Egger. Performance modeling of parallel loops on multi-socket platforms using queueing systems. *IEEE Transactions on Parallel and Distributed Systems*, 31(2):318–331, 2020.

[8] Alexandre Denis, Emmanuel Jeannot, and Philippe Swartvagher. Interferences between Communications and Computations in Distributed HPC Systems. In *ICPP 2021 - 50th International Conference on Parallel Processing*, page 11, Chicago / Virtual, United States, August 2021.

[9] Alexandre Denis, Emmanuel Jeannot, and Philippe Swartvagher. Modeling Memory Contention between Communications and Computations in Distributed HPC Systems. In *IPDPS - 2022 - IEEE International Parallel and Distributed Processing Symposium Workshops*, page 10, Lyon / Virtual, France, May 2022.

[10] Daniel Goodman, Georgios Varisteas, and Tim Harris. Pandia: Comprehensive contention-sensitive thread placement. In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, page 254–269, New York, NY, USA, 2017. Association for Computing Machinery.

[11] Georg Hager, Gerald Schubert, Thomas Schoenemeyer, and Gerhard Wellein. Prospects for truly asynchronous communication with pure MPI and hybrid MPI/OpenMP on current supercomputing platforms. 01 2011.

[12] T. Hoefler and A. Lumsdaine. Message progression in parallel computing - to thread or not to thread? In *2008 IEEE International Conference on Cluster Computing*, pages 213–222, Sep. 2008.

[13] J. Langguth, X. Cai, and M. Sourouri. Memory Bandwidth Contention: Communication vs Computation Tradeoffs in Supercomputers with Multicore Architectures. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 497–506, 2018.

[14] Zoltan Majo and Thomas R. Gross. Memory system performance in a numa multicore multiprocessor. In *Proceedings of the 4th Annual International Conference on Systems and Storage*, SYSTOR '11, New York, NY, USA, 2011. Association for Computing Machinery.

[15] Bogdan Marius Tudor, Yong Meng Teo, and Simon See. Understanding off-chip memory contention of parallel programs in multicore systems. In *2011 International Conference on Parallel Processing*, pages 602–611, 2011.

[16] Wei Wang, Jack W. Davidson, and Mary Lou Soffa. Predicting the memory bandwidth and optimal core allocations for multi-threaded applications on large-scale numa machines. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 419–431, 2016.

A Algorithms

Algorithm 1 Predict total memory bandwidth (implementation of equation 2)

Inputs: N_{par}^{max}, T_{par}^{max}, N_{seq}^{max}, N_{par}^{max}, T_{par}^{max2}, δ_l, δ_r
Output: $\mathcal{T}$

```
function PredictTotal
    𝒯 ← []
    for i = 1 to number_cores do
        if i ≤ N_par^max then
            𝒯[i] ← T_par^max
        else if i ≤ N_seq^max then
            𝒯[i] ← T_par^max − δ_l × (i − N_par^max)
        else
            𝒯[i] ← T_par^max2 − δ_r × (i − N_seq^max)
        end if
    end for
end function
```

Algorithm 2 Compute memory bandwidths available for computations and communications (implementation of equations 4, 5, 6, 7 and 8)

Inputs: $\mathcal{T}$, B_{seq}^{comp}, T_{seq}^{max}, N_{seq}^{max}, N_{par}^{max}, B_{seq}^{comm}, α
Outputs: $\mathcal{B}_{par}^{comp}$, $\mathcal{B}_{par}^{comm}$, $\mathcal{B}_{seq}^{comp}$

1: **function** PredictBandwidths
2: $\quad \mathcal{B}_{par}^{comm} \leftarrow []$
3: $\quad \mathcal{B}_{par}^{comp} \leftarrow []$
4: $\quad \mathcal{B}_{seq}^{comp} \leftarrow []$
5: $\quad \beta_v \leftarrow None$
6: $\quad \beta_i \leftarrow None$
7: $\quad$ **for** $i = 1$ **to** $number_cores$ **do**
8: $\quad\quad \mathcal{B}_{seq}^{comp}[i] \leftarrow \min(B_{seq}^{comp} \times i, \mathcal{T}[i], T_{seq}^{max})$ ▷ Equation 8
9: $\quad\quad$ **if** $i \times B_{seq}^{comp} + \alpha \times B_{seq}^{comm} < \mathcal{T}[i]$ **then** ▷ Equation 4
10: $\quad\quad\quad \mathcal{B}_{par}^{comp}[i] \leftarrow i \times B_{seq}^{comp}$ ▷ Equation 5
11: $\quad\quad\quad \mathcal{B}_{par}^{comm}[i] \leftarrow \min(\mathcal{T}[i] - \mathcal{B}_{par}^{comp}[i], B_{seq}^{comm})$ ▷ Equation 6
12: $\quad\quad\quad \beta_v \leftarrow \mathcal{B}_{par}^{comm}[i]/B_{seq}^{comm}$
13: $\quad\quad\quad \beta_i \leftarrow i$
14: $\quad\quad$ **else**
15: $\quad\quad\quad \alpha_i \leftarrow \alpha$ ▷ Equation 7
16: $\quad\quad\quad$ **if** $(N_{seq}^{max} - N_{par}^{max}) > 1$ **and** $i < N_{seq}^{max}$ **then**
17: $\quad\quad\quad\quad \delta_c \leftarrow (\beta_v - \alpha)/(N_{seq}^{max} - \beta_i)$
18: $\quad\quad\quad\quad \alpha_i \leftarrow \beta_v - \delta_c \times (i - \beta_i)$
19: $\quad\quad\quad$ **end if**
20: $\quad\quad\quad \mathcal{B}_{par}^{comm}[i] \leftarrow \alpha_i \times B_{seq}^{comm}$ ▷ Equation 6
21: $\quad\quad\quad \mathcal{B}_{par}^{comp}[i] \leftarrow \mathcal{T}[i] - \mathcal{B}_{par}^{comm}[i]$ ▷ Equation 5
22: $\quad\quad$ **end if**
23: $\quad$ **end for**
24: **end function**

Algorithm 3 Predict communication performances according to memory placements (implementation of equation 9)

Inputs: m_{comp}, m_{comm}, $\mathcal{M}_{local}$, $\mathcal{M}_{remote}$
Outputs: $\mathcal{B}_{par}^{comm}$

1: **function** GetCommBandwidths
2: $\quad$ **if** $m_{comp} == m_{comm}$ **and** $m_{comp} \geq \#m$ **then**
3: $\quad\quad \mathcal{B}_{par}^{comp}, \mathcal{B}_{par}^{comm}, \mathcal{B}_{seq}^{comp} \leftarrow$ PredictBandwidths($\mathcal{M}_{remote}$)
4: $\quad\quad$ **return** $\mathcal{B}_{par}^{comm}$
5: $\quad$ **else**
6: $\quad\quad$ **if** $m_{comm} \geq \#m$ **then**
7: $\quad\quad\quad \mathcal{B}_{par}^{comp}, \mathcal{B}_{par}^{comm}, \mathcal{B}_{seq}^{comp} \leftarrow$ PredictBandwidths(
$\quad\quad\quad\quad \mathcal{M}_{local}$,
$\quad\quad\quad\quad B_{seq}^{comm} \leftarrow B_{seq}^{comm}(\mathcal{M}_{remote})$ ▷ Use B_{seq}^{comm} from $\mathcal{M}_{remote}$ in the function
$\quad\quad\quad$)
8: $\quad\quad\quad$ **return** $\mathcal{B}_{par}^{comm}$
9: $\quad\quad$ **else**
10: $\quad\quad\quad \mathcal{B}_{par}^{comp}, \mathcal{B}_{par}^{comm}, \mathcal{B}_{seq}^{comp} \leftarrow$ PredictBandwidths($\mathcal{M}_{local}$)
11: $\quad\quad\quad$ **return** $\mathcal{B}_{par}^{comm}$
12: $\quad\quad$ **end if**
13: $\quad$ **end if**
14: **end function**

Algorithm 4 Predict computation performances according to memory placements (implementation of equation 10)

Inputs: $m_{comp},\ m_{comm},\ \mathcal{M}_{local},\ \mathcal{M}_{remote}$
Outputs: $\mathcal{B}_{par}^{comp}$

```
function GetCompBandwidths
    if m_comp < #m then
        B_par^comp, B_par^comm, B_seq^comp ← PredictBandwidths(M_local)
        if m_comp == m_comm then
            return B_par^comp
        else
            return B_seq^comp
        end if
    else
        B_par^comp, B_par^comm, B_seq^comp ← PredictBandwidths(M_remote)
        if m_comp == m_comm then
            return B_par^comp
        else
            return B_seq^comp
        end if
    end if
end function
```

B Model parameter values

Parameter		$\mathcal{M}_{local}$	$\mathcal{M}_{remote}$
N_{par}^{max}	(number of computing cores)	32	32
T_{par}^{max}	(MB/s)	95555.5	75276.9
N_{seq}^{max}	(number of computing cores)	32	32
T_{seq}^{max}	(MB/s)	84420.4	76466.4
T_{par}^{max2}	(MB/s)	95555.5	75276.9
α		0.842	0.593
δ_l	(MB/s/core)	0.0	0.0
δ_r	(MB/s/core)	0.0	0.0
B_{seq}^{comp}	(MB/s)	2808.8	2746.5
B_{seq}^{comm}	(MB/s)	12793.0	18898.9

Table 5: Parameter values for executions on `billy`

Parameter		$\mathcal{M}_{local}$	$\mathcal{M}_{remote}$
N_{par}^{max}	(number of computing cores)	18	8
T_{par}^{max}	(MB/s)	83849.6	32315.3
N_{seq}^{max}	(number of computing cores)	18	8
T_{seq}^{max}	(MB/s)	80107.7	32733.3
T_{par}^{max2}	(MB/s)	83849.6	32315.3
α		0.951	0.936
δ_l	(MB/s/core)	0.0	0.0
δ_r	(MB/s/core)	0.0	1.6
B_{seq}^{comp}	(MB/s)	4489.2	4487.6
B_{seq}^{comm}	(MB/s)	9948.4	8784.7

Table 6: Parameter values for executions on `bora`

Parameter		$\mathcal{M}_{local}$	$\mathcal{M}_{remote}$
N_{par}^{max}	(number of computing cores)	11	5
T_{par}^{max}	(MB/s)	72147.6	32102.5
N_{seq}^{max}	(number of computing cores)	14	5
T_{seq}^{max}	(MB/s)	70072.9	32677.4
T_{par}^{max2}	(MB/s)	71509.2	32102.5
α		0.959	0.949
δ_l	(MB/s/core)	212.8	0.0
δ_r	(MB/s/core)	656.8	-38.4
B_{seq}^{comp}	(MB/s)	6656.5	7171.3
B_{seq}^{comm}	(MB/s)	11341.2	10607.0

Table 7: Parameter values for executions on `dahu`

Parameter		$\mathcal{M}_{local}$	$\mathcal{M}_{remote}$
N_{par}^{max}	(number of computing cores)	32	32
T_{par}^{max}	(MB/s)	81846.8	87081.5
N_{seq}^{max}	(number of computing cores)	32	32
T_{seq}^{max}	(MB/s)	70092.1	67694.6
T_{par}^{max2}	(MB/s)	81846.8	87081.5
α		0.967	0.908
δ_l	(MB/s/core)	0.0	0.0
δ_r	(MB/s/core)	0.0	0.0
B_{seq}^{comp}	(MB/s)	2221.1	2210.2
B_{seq}^{comm}	(MB/s)	12139.7	22394.3

Table 8: Parameter values for executions on `diablo`

Parameter		$\mathcal{M}_{local}$	$\mathcal{M}_{remote}$
N_{par}^{max}	(number of computing cores)	14	5
T_{par}^{max}	(MB/s)	75015.2	31928.2
N_{seq}^{max}	(number of computing cores)	15	5
T_{seq}^{max}	(MB/s)	73818.4	32576.8
T_{par}^{max2}	(MB/s)	74398.4	31928.2
α		0.953	0.971
δ_l	(MB/s/core)	616.8	0.0
δ_r	(MB/s/core)	649.2	-10.6
B_{seq}^{comp}	(MB/s)	6698.7	7178.5
B_{seq}^{comm}	(MB/s)	9628.8	9345.7

Table 9: Parameter values for executions on `grvingt`

Parameter		$\mathcal{M}_{local}$	$\mathcal{M}_{remote}$
N_{par}^{max}	(number of computing cores)	17	5
T_{par}^{max}	(MB/s)	73423.0	31629.7
N_{seq}^{max}	(number of computing cores)	18	7
T_{seq}^{max}	(MB/s)	72589.9	29130.7
T_{par}^{max2}	(MB/s)	73387.7	31278.1
α		0.915	0.761
δ_l	(MB/s/core)	35.3	175.8
δ_r	(MB/s/core)	0.0	119.8
B_{seq}^{comp}	(MB/s)	4455.4	4455.2
B_{seq}^{comm}	(MB/s)	11481.1	11459.6

Table 10: Parameter values for executions on `henri`

Parameter		$\mathcal{M}_{local}$	$\mathcal{M}_{remote}$
N_{par}^{max}	(number of computing cores)	8	11
T_{par}^{max}	(MB/s)	42487.7	16936.1
N_{seq}^{max}	(number of computing cores)	11	4
T_{seq}^{max}	(MB/s)	43655.6	14726.2
T_{par}^{max2}	(MB/s)	39718.9	15217.8
α		0.853	0.270
δ_l	(MB/s/core)	922.9	859.1
δ_r	(MB/s/core)	191.7	103.3
B_{seq}^{comp}	(MB/s)	4456.4	4455.4
B_{seq}^{comm}	(MB/s)	11450.4	11410.0

Table 11: Parameter values for executions on `henri-subnuma`

Parameter		$\mathcal{M}_{local}$	$\mathcal{M}_{remote}$
N_{par}^{max}	(number of computing cores)	14	7
T_{par}^{max}	(MB/s)	53948.2	24706.4
N_{seq}^{max}	(number of computing cores)	14	7
T_{seq}^{max}	(MB/s)	47817.2	21137.3
T_{par}^{max2}	(MB/s)	53948.2	24706.4
α		1.000	1.000
δ_l	(MB/s/core)	0.0	0.0
δ_r	(MB/s/core)	0.0	39.9
B_{seq}^{comp}	(MB/s)	3417.6	3417.8
B_{seq}^{comm}	(MB/s)	6220.0	6219.5

Table 12: Parameter values for executions on `occigen`

Parameter		$\mathcal{M}_{local}$	$\mathcal{M}_{remote}$
N_{par}^{max}	(number of computing cores)	24	25
T_{par}^{max}	(MB/s)	64626.9	36737.2
N_{seq}^{max}	(number of computing cores)	26	31
T_{seq}^{max}	(MB/s)	62462.7	32649.8
T_{par}^{max2}	(MB/s)	64566.2	36696.7
α		0.990	0.945
δ_l	(MB/s/core)	30.3	6.8
δ_r	(MB/s/core)	56.9	-8.4
B_{seq}^{comp}	(MB/s)	3211.5	3030.8
B_{seq}^{comm}	(MB/s)	4958.6	4390.1

Table 13: Parameter values for executions on `pyxis`