Detailed Modeling of Heterogeneous and Contention-Constrained Point-to-Point MPI Communication

Andreas Thune, Sven-Arne Reinemo, *Senior Member, IEEE*, Tor Skeie, and Xing Cai

***Abstract*—The network topology of modern parallel computing systems is inherently heterogeneous, with a variety of latency and bandwidth values. Moreover, contention for the bandwidth can exist on different levels when many processes communicate with each other. Many-pair, point-to-point MPI communication is thus characterized by heterogeneity and contention, even on a cluster of homogeneous multicore CPU nodes. To get a detailed understanding of the individual communication cost per MPI process, we propose a new modeling methodology that incorporates both heterogeneity and contention. First, we improve the standard max-rate model to better quantify the actually achievable bandwidth depending on the number of MPI processes in competition. Then, we make a further extension that more detailedly models the bandwidth contention when the competing MPI processes have different numbers of neighbors, with also non-uniform message sizes. Thereafter, we include more flexibility by considering interactions between intra-socket and inter-socket messaging. Through a series of experiments done on different processor architectures, we show that the new heterogeneous and contention-constrained performance models can adequately explain the individual communication cost associated with each MPI process. The largest test of realistic point-to-point MPI communication involves 8,192 processes and in total 2,744,632 simultaneous messages over 64 dual-socket AMD Epyc Rome compute nodes connected by InfiniBand, for which the overall prediction accuracy achieved is 84%.**

***Index Terms*—Intra-node communication, performance modeling, point-to-point MPI communication.**

I. Introduction

MODERN platforms of parallel computing are heterogeneous at least with respect to the interconnect. Even on a system purely based on multicore CPUs, the connectivity between the CPU cores has several layers. The cores that belong to the same CPU socket can communicate more efficiently than those between sockets, because the inter-socket memory bandwidth is lower than the intra-socket counterpart. At the cluster level, between any pair of compute nodes, the communication speed is even lower and can depend on the actual location of the nodes on the network setup. All these levels of interconnect heterogeneity will translate into vastly different values of the effective latency and bandwidth of point-to-point MPI communication.

Another complicating factor for many-pair, point-to-point MPI communication on today's parallel platforms is the potential competition between different MPI processes. This is because each CPU socket can, if needed, support a large number of concurrent MPI processes. Contention arises when multiple pairs of sending-receiving processes simultaneously communicate over the same connection. Such contention may exist, in different magnitudes, over the entire network. Moreover, the competition situation is often *dynamically* changing; for instance, an MPI process pair communicating a small message may complete before the other MPI process pairs, resulting in a reduced level of contention.

This paper aims to detailedly model the per-process overhead associated with realistic, many-pair, point-to-point MPI communication. We want to improve the state-of-the-art quantitative models of point-to-point MPI communication by a new methodology for modeling the bandwidth contention due to a large number of MPI processes that compete against each other. Here, communication can exist on different levels: intra-socket, inter-socket and inter-node. In addition, we target the real-world situation where different MPI processes can have different numbers of neighbors to exchange data with, while the size of each message is also highly non-uniform. One typical example of such a heterogeneous scenario arises from numerically solving a partial differential equation (PDE) over an irregular solution domain. The first step of parallelizing a mesh-based PDE solver is to partition an unstructured computational mesh, which covers the irregular solution domain, into a desirable number of subdomains each assigned to an MPI process. The usual partitioning result is that the number of the nearest neighbors varies from process to process, and so does the size of each MPI message.

Specifically, we will propose a new modeling methodology that quantifies both heterogeneity and contention. At the same

Manuscript received 7 July 2022; revised 27 February 2023; accepted 2 March 2023. Date of publication 8 March 2023; date of current version 24 March 2023. The work of Andreas Thune was supported by the Research Council of Norway under Grant 237898. The work of Xing Cai was supported in part by the European High-Performance Computing Joint Undertaking under Grant 956213, and in part by the Research Council of Norway under Grant 329017. The research presented in this paper benefited from the Experimental Infrastructure for the Exploration of Exascale Computing (eX3), which was financially supported in part by the Research Council of Norway under Grant 270053, and in part by resources provided by Sigma2 - the National Infrastructure for High Performance Computing and Data Storage in Norway . Recommended for acceptance by S. Chandrasekaran. *(Corresponding author: Andreas Thune.)*

Andreas Thune, Tor Skeie, and Xing Cai are with the Simula Research Laboratory, 0164 Oslo, Norway (e-mail: andreast@simula.no; tskeie@simula.no; xingca@simula.no).

Sven-Arne Reinemo is with the Simula Metropolitan Centre for Digital Engineering, 0167 Oslo, Norway (e-mail: svenar@simula.no).

This article has supplementary downloadable material available at https://doi.org/10.1109/TPDS.2023.3253881, provided by the authors.

Digital Object Identifier 10.1109/TPDS.2023.3253881

time, we want to inherit a level of simplicity from the fundamental *postal* model [1], [2] that describes a single pair of MPI processes, and the successor max-rate model [3]. The elegantly simple postal model relies on only two parameters to quantify the cost of point-to-point communication, i.e., a start-up latency τ and a bandwidth value BW. The max-rate model lets BW depend linearly on the number of competing MPI processes while limited from above by a maximum bandwidth value, which is prescribed as the third parameter. Our new performance models are also based on a fair competition among the MPI processes, but the value of BW will depend *dynamically* on the actual number of competing MPI processes and how these processes affect each other across two specific levels: intra-socket and inter-socket. We focus our modeling and experiments on large messages, that use the rendezvous MPI protocol. The contributions of our work are as follows:

- We extend the `osu_bibw` micro-benchmark of MVAPICH [4] to easily tabulate the various τ and BW values, both depend on the connection type and the latter is also a function of the number of competing MPI processes. These tabulated values serve as a characterization of the communication performance of a heterogeneous interconnect.
- We improve the accuracy of the max-rate model [3] for the case of multiple MPI process pairs concurrently exchanging equal-sized messages. Specifically, the tabulated BW values replace an often idealized relationship between the achievable bandwidth and the number of competing MPI pairs.
- We propose a "staircase" strategy to detail the contention between many MPI processes with varying numbers of neighbors and message sizes, when competing over a single level of interconnect.
- We extend the single-level "staircase" modeling to mixed-level "staircase" modeling that also quantifies the interaction between two particular communication levels: intra-socket and inter-socket.

The remainder of the paper is organized as follows. Section II introduces a new bi-directional multi-pair micro-benchmark, which can be used to pinpoint the achievable bandwidth as a function of the competing send-receive pairs. Section III is devoted to a new "staircase" modeling strategy that can be adopted to handle the various types of heterogeneity, more specifically, non-uniform message size, a varied number of messages per MPI process, and the interaction between intra-socket and inter-socket communication. Section IV tests the "staircase" strategy and the resulting new models in realistic cases of many-pair, point-to-point MPI communication. Section V places our current work with respect to the existing work on modeling MPI point-to-point communication, whereas Section VI provides some concluding remarks. The source code that implements the mixed-level modeling strategy can be found in the appendix, available online.

II. Detailing Bandwidth Contention

As mentioned above, the postal model [1], [2] provides an elegant and simple way of quantifying the time needed to pass a message between a single pair of MPI send-receive processes. Its formula is as follows:

$$T(s) = \tau + \frac{s}{BW_{\text{SP}}}, \tag{1}$$

where the constant parameter τ denotes the start-up latency, s denotes the size of the MPI message, and the constant parameter BW_{SP} denotes the communication bandwidth. The subscript "SP" stands for single-pair and thus highlights the applicability of the postal model. Experiments (see e.g. [5]) have shown that this two-parameter model can produce estimates of $T(s)$ that agree very well with the actual single-message time usages, as long as the message size s is within the regime of the same protocol (i.e., short, eager, or rendezvous). It also means that each protocol is associated with its specific set of τ and BW_{SP} parameters.

The max-rate model [3] extends the postal model by considering N competing MPI messages belonging to N send-receive process pairs. If all the messages are of the same size s, they will have the same time usage due to a fair competition for the bandwidth. The simplest formula of the max-rate model is as follows:

$$T(N, s) = \tau + \frac{N \cdot s}{BW_{\text{MP}}(N)}. \tag{2}$$

In the above formula, $BW_{\text{MP}}(N)$ is meant to model a shared bandwidth to be fairly competed among the N messages, and the subscript "MP" stands for multi-pair. The dependency of BW_{MP} on N is considered by the max-rate model in its simplest form as follows:

$$BW_{\text{MP}}(N) = \min\left(N \cdot BW_{\text{SP}}, BW_{\max}\right), \tag{3}$$

where $BW_{\max}$ denotes the upper limit of the achievable communication bandwidth, i.e., a *max rate*. The existence of $BW_{\max}$ illustrates a saturation effect, which applies to both communication over a network connection and communication through shared memory.

An extended max-rate model was presented in [6] to consider variations in the message size and number of messages per process. It mainly targets inter-node communication

$$T = M \cdot \tau + \max\left(\frac{s_{\text{total}}}{BW_{\max}}, \frac{s_{\max}}{BW_{\text{SP}}}\right). \tag{4}$$

Here, M is the maximum number of messages per process, s_{total} and $s_{\max}$ denote, respectively, the total messaging volume per node and the maximum per-process volume. We note that (4) only models the slowest process per node.

While the max-rate model is simple to use, it has several weaknesses. First, the actual $BW_{\text{MP}}(N)$ value may not be linearly proportional to N before hitting the upper limit $BW_{\max}$, especially for intra- and inter-socket traffic. Second, the bandwidth contention may not be accurately modeled when the processes have largely varying message numbers and/or sizes. Third, only the slowest process is modeled by (4), not the earlier finishing processes.

In the remainder of this section we will improve the max-rate model with respect to its first shortcoming. This will be achieved

Algorithm 1: Bi-Directional Multi-Pair Benchmark.

```
1: $P$, initial_size, max_size, increase_factor, num_repeats
2: $size$ = initial_size
3: while $size <$ max_size do
4:   if rank $< \frac{P}{2}$ then
5:     for $i = 1, \ldots,$ num_repeats do
6:       MPI_Isend(send_data_buffer + $i \cdot size$, $size$, $rank + \frac{P}{2}$)
7:       MPI_Irecv(recv_data_buffer + $i \cdot size$, $size$, $rank + \frac{P}{2}$)
8:     end for
9:     MPI_Waitall()
10:  else
11:    for $i = 1, \ldots,$ num_repeats do
12:      MPI_Irecv(recv_data_buffer + $i \cdot size$, $size$, $rank - \frac{P}{2}$)
13:      MPI_Isend(send_data_buffer + $i \cdot size$, $size$, $rank - \frac{P}{2}$)
14:    end for
15:    MPI_Waitall()
16:  end if
17:  $size$ = increase_factor $\cdot$ $size$
18: end while
```

by extending a micro-benchmark of MVAPICH [4]. The other two shortcomings will be addressed in Section III.

A. A New Micro-Benchmark

We want to pinpoint the actual BW_{MP} values through measurements obtained by a simple benchmark, instead of using the often idealized formula (3). Specifically, we will adopt a new "bi-directional multi-pair" micro-benchmark, as described in Algorithm 1. It is a modification of the `osu_bibw` benchmark of MVAPICH [4]. The new micro-benchmark involves P processes to form $\frac{P}{2}$ sender-receiver pairs, each simultaneously handling two equal-sized messages of opposite directions. The total number of competing messages is thus P. To avoid the unwanted side-effect of MPI messages being cached, each repetition of a message is loaded/stored from/to a different location in a pre-allocated long buffer.

B. Example: Measuring $BW_{MP}(N)$ on Four machines

We have run the bi-directional multi-pair micro-benchmark (Algorithm 1) on four machines of dual-socket CPUs. The specific CPU types are: (1) ARM Cavium 32-core ThunderX2 CN9980; (2) ARM 64-core Kunpeng 920-6426; (3) Intel Xeon 26-core Gold-6230; and (4) AMD 64-core Epyc Rome-7742. OpenMPI v4.0.5 was used as the MPI installation. Compilation used GCC v10.1.0 with the `-O3 -march=armv8-a` options on the ThunderX2 and Kunpeng machines, and GCC v10.2.0 with the `-O3` option on the Intel Xeon and Epyc Rome machines. On each machine, we measure the time usage for a series of message sizes in the regime of the rendezvous protocol.[1] This is repeated for some chosen numbers of MPI processes. For each chosen value N, the series of time measurements (for the different message sizes) undergo a linear regression to recover the values of τ and $BW_{\mathrm{MP}}(N)$ that can be used in the max-rate model (2) or later in our new models to be introduced in Section III. On a given machine and for a specific communication level, the recovered τ values are very similar for the different choices of N. So we have consistently used the τ value associated with $N = 2$ in Table I and later experiments. We remark that the models and experiments to be presented in this paper target message sizes in the regime of the rendezvous protocol. Our modeling approach remains the same for the other protocols.

The measurement-determined $BW_{\mathrm{MP}}(N)$ values for all the four machines are summarized in Table I, where we also distinguish the scenarios of intra-socket and inter-socket. The former means that all sender-receiver pairs are placed on the same socket, whereas the latter means that each pair is split across two sockets. In this table, and also in the remainder of the paper, the value N denotes the number of active MPI processes per socket that concurrently receive incoming messages, i.e., running the micro-benchmark of Algorithm 1 with $P = N$ for the intra-socket measurements and $P = 2\,N$ for the inter-socket measurements. The reason for this definition of N is because the new performance models to be introduced in Section III consider a socket as the "base unit" when modeling intra- and inter-socket communications. The $BW_{\max}$ values in Table I are obtained from running the same number of MPI processes per socket as the CPU cores. The intra-socket BW_{SP} value is obtained by running two MPI processes on just one socket, with only one uni-directional message.

One clear observation from Table I is that the τ measurements associated with the intra-socket and inter-socket cases confirm that intra-socket MPI communication is faster than the inter-socket counterpart. Another important observation is that the intra- and inter-socket $BW_{\mathrm{MP}}(N)$ values do not follow the formula (3) of the max-rate model, clearly demonstrated in Fig. 1. These tabulated $BW_{\mathrm{MP}}(N)$ values will be heavily used in our new models of Section III that are capable of handling varying numbers of incoming/outgoing messages per MPI process, non-uniform size per message, and the interaction between intra- and inter-socket traffic.

III. New Performance Models of Many-Pair, Point-to-Point Communication

Apart from replacing the max-rate formula (3) with tabulated $BW_{\mathrm{MP}}(N)$ values that are determined by the new micro-benchmark of Algorithm 1, another improvement is to more accurately model the bandwidth contention when the competing MPI processes receive largely different volumes. We will thus develop in this section a new modeling strategy that detailedly

[1]Time measurements of messages of size in the regime of the short or eager protocol will produce another set of τ and $BW_{\mathrm{MP}}(N)$ values.

TABLE I
VALUES OF τ (IN μS) AND $BW_{\mathrm{MP}}(N)$ (IN GB/S), OBTAINED FROM A LINEAR REGRESSION OF THE TIME MEASUREMENTS OF THE BI-DIRECTIONAL MULTI-PAIR MICRO-BENCHMARK (ALGORITHM 1) ON FOUR DUAL-SOCKET CPU MACHINES. THE TABULATED $BW_{\mathrm{MP}}(N)$ VALUES WILL IMPROVE THE ACCURACY OF THE MAX-RATE MODEL (2)-(3) FOR INTRA- AND INTER-SOCKET COMMUNICATION

CPU type	Level	τ	BW_{SP}	$BW_{\mathrm{MP}}(2)$	$BW_{\mathrm{MP}}(4)$	$BW_{\mathrm{MP}}(8)$	$BW_{\mathrm{MP}}(16)$	$BW_{\max}$
ThunderX2 CN9980	intra-socket	2.3	7.5	14.6	25.5	32.5	45.0	54.0
	inter-socket	4.4	6.5	13.7	17.8	18.3	19.7	22.7
Kunpeng 920-6426	intra-socket	3.8	5.0	10.5	14.8	17.0	19.7	53.7
	inter-socket	5.1	4.6	9.1	10.4	13.3	18.6	49.5
Intel Xeon Gold 6230	intra-socket	1.6	10.0	15.9	23.8	31.7	42.7	42.7
	inter-socket	2.7	8.1	15.6	22.5	27.5	29.4	29.4
AMD Epyc 7742 Rome	intra-socket	1.7	10.2	16.8	17.6	19.2	23.4	51.0
	inter-socket	2.9	5.3	8.7	11.1	12.2	13.0	30.3

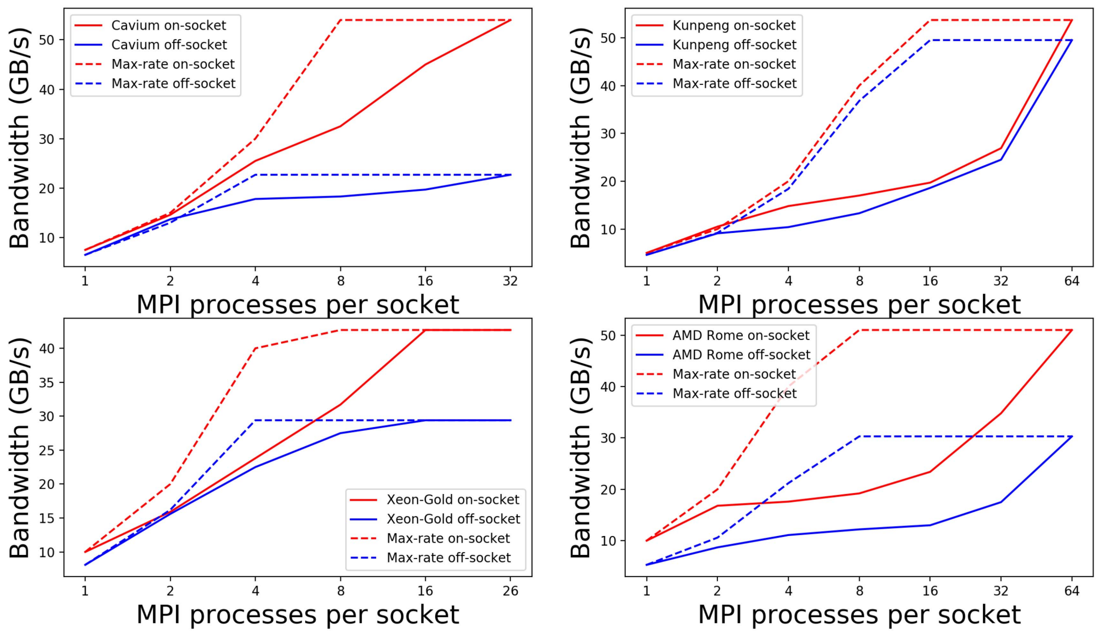
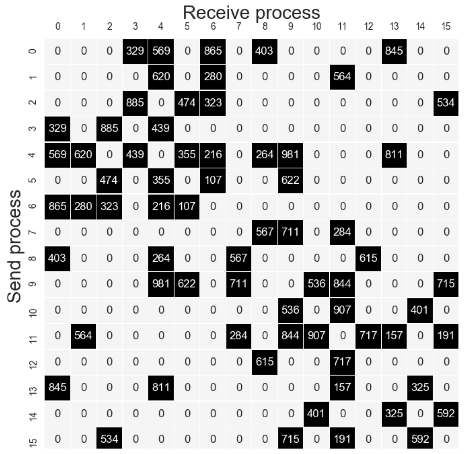


Fig. 1. Comparison between measurement-pinpointed $BW_{\mathrm{MP}}(N)$ values (solid curves) and those suggested by the formula (3) of the max-rate model (dashed curves). The measurements are obtained on four dual-socket CPU machines: ARM Cavium ThunderX2 (top-left), ARM Kunpeng (top-right), Intel Xeon-Gold (bottom-left) and AMD Epyc Rome (bottom-right).

quantify the *per-process* time usage for the cases where the message size is non-uniform and/or the number of neighbors per process varies. A more general situation is that many MPI messages of various sizes are concurrently communicated over multiple levels of a heterogeneous network. In total three new models of increasing complexity will be introduced.

A. *General Many-Pair, Point-to-Point Communication*

First let us define the general situation of many-pair, point-to-point MPI communication in Algorithm 2. More specifically, MPI process with rank i will simultaneously receive M_i^{in} messages from M_i^{in} different neighbors. Each process thus has a list of neighbor identities $\{\mathrm{neigh}_j^{\mathrm{in}}\}_{j=0}^{M_i^{\mathrm{in}}-1}$. The neighbors can be of mixed types, i.e., some may reside on the same socket as process i, others may reside on a different socket but inside the same compute node, whereas the rest may reside on other compute nodes. Each process is also assumed to send M_i^{out} outgoing messages to M_i^{out} neighbors (can be different from the M_i^{in} "inward" neighbors). All the messages can be of varying sizes, and the sizes of the "inward" and "outward" messages do not have to match. For this purpose, we have for each process two sets of message sizes $\{\mathrm{Rsize}_j\}_{j=0}^{M_i^{\mathrm{in}}-1}$ and $\{\mathrm{Ssize}_j\}_{j=0}^{M_i^{\mathrm{out}}-1}$. An example of the resulting sparse *communication matrix* is Fig. 2, which involves 16 processes. It should be remarked that Algorithm 2 describes irregular point-to-point MPI communications that frequently arise in scientific computations. One such example can be found in the `copyOwnerToAll` function of the DUNE software framework [7], [8], [9], in connection with parallelizing sparse matrix-vector multiplications.


Receive process
Send process

Send \ Receive	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	329	569	0	865	0	403	0	0	0	0	845	0	0
1	0	0	0	0	620	0	280	0	0	0	0	564	0	0	0	0
2	0	0	0	885	0	474	323	0	0	0	0	0	0	0	0	534
3	329	0	885	0	439	0	0	0	0	0	0	0	0	0	0	0
4	569	620	0	439	0	355	216	0	264	981	0	0	0	811	0	0
5	0	0	474	0	355	0	107	0	0	622	0	0	0	0	0	0
6	865	280	323	0	216	107	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	567	711	0	284	0	0	0	0
8	403	0	0	0	264	0	0	567	0	0	0	0	615	0	0	0
9	0	0	0	0	981	622	0	711	0	0	536	844	0	0	0	715
10	0	0	0	0	0	0	0	0	0	536	0	907	0	0	401	0
11	0	564	0	0	0	0	0	284	0	844	907	0	717	157	0	191
12	0	0	0	0	0	0	0	0	615	0	0	717	0	0	0	0
13	845	0	0	0	811	0	0	0	0	0	0	157	0	0	325	0
14	0	0	0	0	0	0	0	0	0	0	401	0	0	325	0	592
15	0	0	534	0	0	0	0	0	0	715	0	191	0	0	592	0

Fig. 2. An example communication matrix involving 16 MPI processes. Each black box represents an MPI message, and the number inside the box is the message size.

Algorithm 2: Many-Pair, Point-to-Point Communication.

```
1:  On each process $i$: $\{\text{neigh}^{\text{in}}_j\}_{j=0}^{M_i^{\text{in}}-1}$, $\{\text{Rsize}_j\}_{j=0}^{M_i^{\text{in}}-1}$, $\{\text{neigh}^{\text{out}}_j\}_{j=0}^{M_i^{\text{out}}-1}$, $\{\text{Ssize}_j\}_{j=0}^{M_i^{\text{out}}-1}$
2:  for $j = 0, \dots, M_i^{\text{out}} - 1$ do
3:      MPI_Isend(Sbuffer$_j$, Ssize$_j$, neigh$^{\text{out}}_j$, send_req$_j$)
4:  end for
5:  for $j = 0, \dots, M_i^{\text{in}} - 1$ do
6:      MPI_Irecv(Rbuffer$_j$, Rsize$_j$, neigh$^{\text{in}}_j$, recv_req$_j$)
7:  end for
8:  for $j = 0, \dots, M_i^{\text{out}} - 1$ do
9:      MPI_Wait(send_req$_j$)
10: end for
11: for $j = 0, \dots, M_i^{\text{in}} - 1$ do
12:     MPI_Wait(recv_req$_j$)
13: end for
```

B. A Staircase Modeling Strategy

Before developing new performance models of increasing complexity for quantifying the per-process time usage associated with Algorithm 2, we need to first introduce a "staircase" modeling strategy. The purpose is to quantitatively describe the *dynamically* changing scenarios of competition, i.e., the number of competing processes and the available communication bandwidth are both dynamic. The following principles are associated with this modeling strategy:

1) The time needed by process i to complete all its communication tasks is determined by either how soon it can finish receiving all its M_i^{in} incoming messages, or how soon all its M_i^{out} outgoing messages are received at the destinations. The maximum of the two time usages will apply. The reason for also considering the delivery of the outgoing messages at the destinations is motivated by the most common situation that uses the rendezvous protocol without intermediate buffering, where experiments show that a sending process cannot complete until all its outgoing messages are delivered.
2) The "expected" order of which the M_i^{in} incoming messages are received by process i is determined by the sizes of these messages. (The actual order is stochastic, thus not modelable.) The smallest incoming message completes first, followed by the second smallest message, and so on. The completion time points for the different incoming messages can be estimated using a *staircase* strategy, which will be detailed by formula (9). If the M_i^{in} incoming messages are of the same size, they are considered to complete simultaneously, due to fairness.
3) When multiple receiving processes, each with one or more incoming messages, compete for the same network connection, another staircase principle applies as will be described in formula (5). The process with the least amount of incoming communication will complete first, letting the remaining processes continue with their remaining communication. This procedure repeats itself until all the processes are completed. At all times, the bandwidth is shared evenly between the still active processes, while the actually available bandwidth can depend on the number of concurrently competing processes.

C. Model for One-Level, Single-Neighbor, Non-Uniform Message Size

The staircase modeling strategy will be first applied to the scenario of each MPI process having exactly one incoming message and one outgoing message. The messages sent/received by the different processes can be of various sizes. For this simple scenario we consider that all the MPI processes communicate on the same level, i.e., either all intra-socket, or all inter-socket, or all inter-node.

To model the per-process time usage, we first group the MPI processes that share the same bandwidth, i.e., the processes that reside on the same socket for the level of intra-socket communication, or the processes on one socket that share the same inter-socket bandwidth, or the processes that reside in one compute node that share the same inter-node bandwidth. We let N denote the number of processes in a group. The bandwidth under contention is characterized by a set of pre-tabulated values of BW_{SP}, $BW_{\text{MP}}(2), \dots$, $BW_{\text{MP}}(N)$, as well as a latency constant τ.

Modeling of the per-process time usage will start with sorting the N processes of each group with respect to the size of the per-process single incoming message: $s_0 \leq s_1 \leq \dots \leq s_{N-1}$. Then, the per-process time usage T_i can be estimated as a "staircase," more specifically

$$t_0^{\text{recv}} = \frac{N \cdot s_0}{BW_{\text{MP}}(N)},$$

$$t_i^{\text{recv}} = t_{i-1}^{\text{recv}} + \frac{(N-i)\cdot(s_i - s_{i-1})}{BW_{\text{MP}}(N-i)}, \quad i = 1, 2, \dots, N-1, \tag{5}$$

$$T_i = \tau + \max\left(t_i^{\text{recv}}, t_i^{\text{send}}\right). \tag{6}$$

The recursive formula in (5) accounts for a decreasing degree of bandwidth contention in the form of a staircase, when more

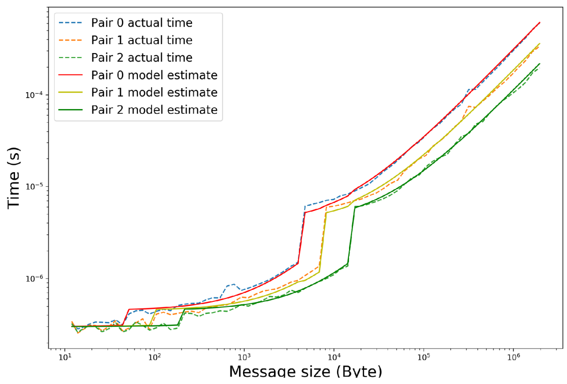


Fig. 3. Single-neighbor, intra-socket messages on an ARM Cavium ThunderX2 CPU. The experiment involves six processes exchanging messages bidirectionally in three pairs. The measured times and model estimates are plotted with dotted and solid lines against the "middle" message size s. Pairs 0, 1 and 2 exchange, respectively, $2\,s$, s and $\frac{s}{2}$ bytes of data.

and more processes complete. Also, we have used $BW_{\text{MP}}(1) = BW_{\text{SP}}$. In (6), t_i^{send} denotes the time needed for the outgoing message of process i to be delivered on the destination process with rank $k = \text{neigh}_i^{\text{out}}$ (see Algorithm 2), which is considered the same as the receiving time t_k^{recv} on process k. We note that the time usage model in (5)-(6) is only valid for single-message per-process communication, but it can be extended to account for multiple message sources and destinations per process. This will be covered in Section III-D below.

C. Verification example: As the verification experiments for this new model, as well as for verifying the other new performance models later, we have always run a benchmark code that implements Algorithm 2. (The total number of MPI processes, the number of neighbors per process and the sizes of the messages may differ from experiment to experiment.) More specifically, each experiment executes Algorithm 2 ten times and the average time usage per iteration is recorded *individually* for each MPI process.

We have tested the new model (5)–(6) by quantifying the per-process time usage of 6 MPI processes running on one ARM Cavium ThunderX2 processor (all the messages are intra-socket). The 6 processes form three sender-receiver pairs, where each pair exchanges the same amount of data in the two opposite directions. However, the three pairs simultaneously work with three different message sizes: $\frac{s}{2}$, s and $2\,s$, with s ranging between 2 bytes and 2 MB. The actual time usages for the three pairs are plotted against the model estimates in Fig. 3.

In Fig. 3 we can see that the model estimates agree very well with the actual time measurements for most of the message sizes. The model (5)–(6) is able to estimate the time usage of each individual process. The two processes of each pair have the same time measurements and estimates, thus we only see three sets of measurement-estimate curves. For each set, we can also observe two clear jumps in the time measurements and estimates, each corresponding to a change in the message protocol, first between the short and eager protocols, then between eager and rendezvous. For example, the eager-rendezvous protocol threshold at 8 KB is clearly visible for the mid-sized pair in the plot.

D. Model for One-Level, Multi-Neighbor, Non-Uniform Message Size

Now let us consider a more general situation where each process communicates with multiple neighbors. For this situation, we modify the recursive formula in (5) by replacing the single-message size s_i with a per-process incoming message volume $V_i = \sum_{j=0}^{M_i^{\text{in}}-1} s_{i,j}$, where M_i^{in} denotes the number of "inward" neighbors for process i, and $s_{i,j}$ denotes the size per incoming message. The MPI processes are organized into groups as in the previous model. Again, N denotes the number of processes in a group, and the processes inside the group is now sorted with respect to $V_0 \le V_1 \le \ldots \le V_{N-1}$. The time usage estimate in (5) for single-neighbor, non-uniform message size communication can now be extended to model the multi-neighbor situation

$$t_0^{\text{recv}} = \frac{N \cdot V_0}{BW_{\text{MP}}(N)},$$
$$t_i^{\text{recv}} = t_{i-1}^{\text{recv}} + \frac{(N-i)\cdot(V_i - V_{i-1})}{BW_{\text{MP}}(N-i)}, \quad i = 1,2,\ldots,N-1, \tag{7}$$

$$T_i = M_i^{\text{in}} \cdot \tau + \max\left(t_i^{\text{recv}}, t_{i,0}^{\text{send}}, t_{i,1}^{\text{send}}, \ldots, t_{i,M_i^{\text{out}}-1}^{\text{send}}\right). \tag{8}$$

In (8), $t_{i,j}^{\text{send}}$ denotes the delivery time needed by each outgoing message, and it equals the time needed by destination process (with rank $\text{neigh}_{i,j}^{\text{out}}$) to receive this message. Since each process can have multiple incoming messages, we thus need to "theoretically" pinpoint the individual time points at which the different messages are received on each destination process. Take for instance process i. It is the destination for M_i^{in} incoming messages, and the total receiving time (without the latency overhead) has been calculated as t_i^{recv} using (7). The following recursive formula, which is again based on a staircase principle, will be used to pinpoint the individual time points at which the M_i^{in} incoming messages are received

$$t_{i,0}^{\text{recv}} = \frac{M_i^{\text{in}} \cdot s_{i,0}}{V_i} \cdot t_i^{\text{recv}},$$
$$t_{i,j}^{\text{recv}} = t_{i,j-1}^{\text{recv}} + \frac{(M_i^{\text{in}} - j)\cdot(s_{i,j} - s_{i,j-1})}{V_i} \cdot t_i^{\text{recv}},$$
$$j = 1,2,\ldots,M_i^{\text{in}} - 1. \tag{9}$$

In the above recursive formula, we have sorted the M_i^{in} incoming messages for process i with respect to the message sizes $s_{i,0} \le s_{i,1} \le \ldots \le s_{i,M_i^{\text{in}}-1}$, where $\sum_{j=0}^{M_i^{\text{in}}-1} s_{i,j} = V_i$. We remark that the completion time point for the largest incoming message, which is theoretically expected to finish last, equals the total receiving time needed by process i, i.e., $t_{i,M_i^{\text{in}}-1}^{\text{recv}} = t_i^{\text{recv}}$.

D. Verification Examples: To demonstrate the single-level, multi-neighbor model (7)-(9), we have conducted three experiments, each consisting exclusively of intra-socket, inter-socket or inter-node traffic. All the results were obtained on a cluster of Epyc Rome CPUs, described in Section II-B. For the intra-socket experiment, a synthetic communication pattern with 64 MPI processes, spread over two sockets, has been used. As shown

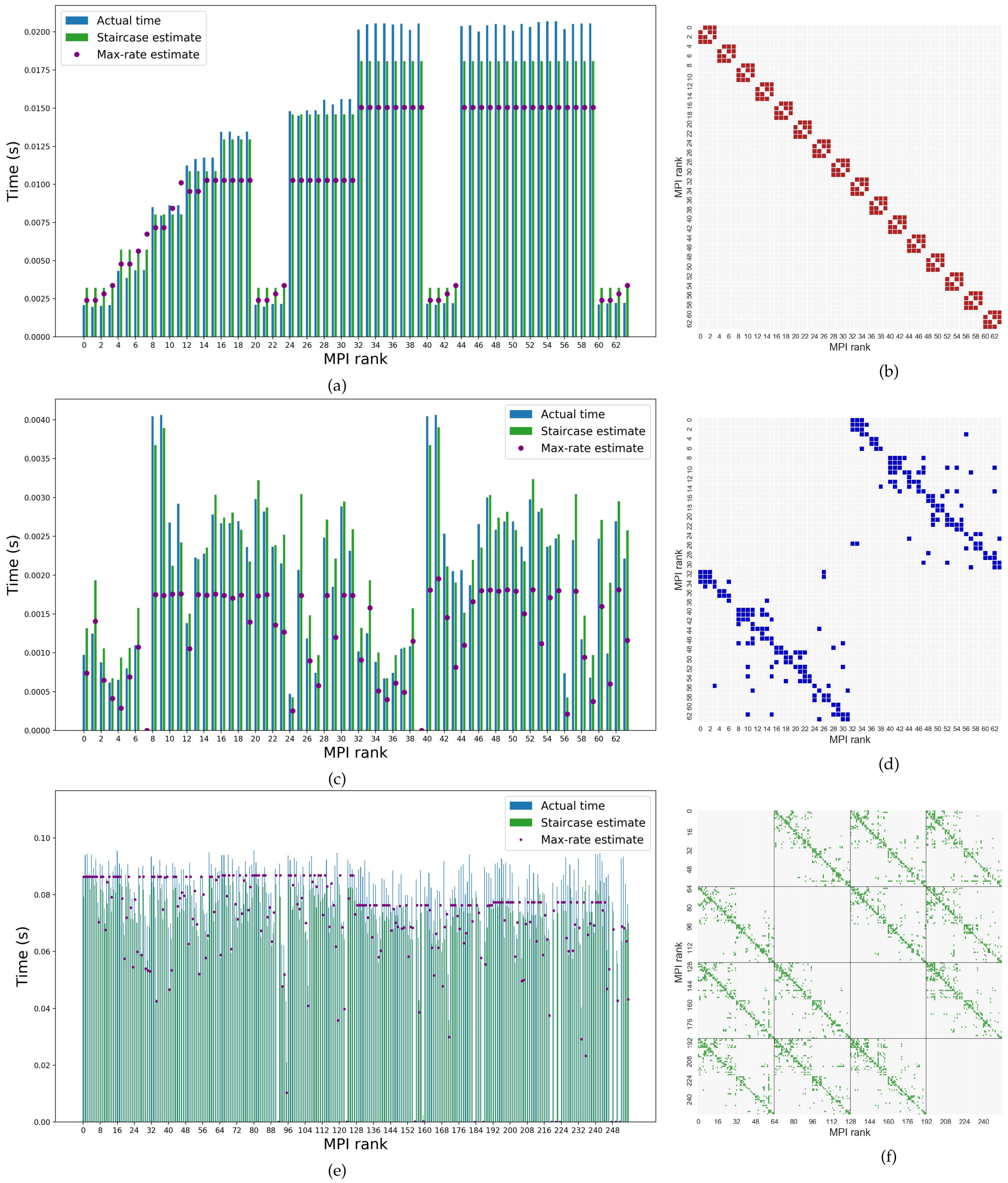


Fig. 4. Actual time measurement (blue bar), staircase model estimate (green bar) and max-rate estimate (purple dots) for each process on Epyc Rome CPUs. The intra-socket experiment (a) uses a synthetic communication pattern shown in (b), while the inter-socket (c) and inter-node (e) results are obtained using realistic communication patterns shown in (d) and (f), respectively.

in Fig. 4(b), each process sends and receives 3 intra-socket messages of different sizes. The per-process time estimates are plotted against the actual time measurements in Fig. 4(a). For comparison, the plot also includes per-process estimates produced by the extended max-rate model (4), marked as purple dots. More specifically, we have used the following variation of (4)

$$T_i = M_i \cdot \tau + \max\left(\frac{\min\left(V_{\text{total}}, NV_i\right)}{BW_{\max}}, \frac{V_i}{BW_{\text{SP}}}\right). \qquad (10)$$

That is, the total messaging volume s_{total} in (4) is replaced by $\min(V_{\text{total}}, NV_i)$. The maximum per-process volume $s_{\max}$ is

TABLE II
INFORMATION ON THE VERIFICATION EXAMPLES PRESENTED IN FIG. 4. THE COLUMNS DISPLAY THE COMMUNICATION TYPE, NUMBER OF MPI PROCESSES, TOTAL NUMBER OF MESSAGES, MAXIMUM/MINIMUM MESSAGES PER PROCESS, TOTAL/MAXIMUM/MINIMUM(NON-ZERO) PER-PROCESS COMMUNICATION VOLUME, AND THE TOTAL RELATIVE PREDICTION ERRORS FOR, RESPECTIVELY, THE STAIRCASE MODEL AND THE EXTENDED MAX-RATE MODEL (10).

Communication type	#procs	#msgs	max/min	total/max/min volume	Staircase error	Max-rate error
intra-socket	64	192	3/3	16.5MB/0.5MB/0.05MB	11.5%	26.0%
inter-socket	64	218	7/0	19.4MB/1.0MB/0.04MB	13.0%	36.8%
inter-node	256	3958	58/0	461.7MB/4.3MB/0.3KB	12.9%	13.0%

replaced by per-process volume V_i. Note that (10) is the same as (4) for the slowest process.

The inter-socket experiment uses 64 MPI processes (also spread over two sockets) that exclusively send and receive inter-socket messages, see Fig. 4(d). The number of neighbors per MPI process and the message sizes arise from partitioning a realistic unstructured computational mesh into 64 pieces, where the maximum number of neighbors per process is 7, and two processes have no neighbors. The message sizes also vary considerably. The actual and estimated per-process times by (7)-(9) are displayed in Fig. 4(c). Max-rate estimates are also included in Fig. 4(c).

The inter-node experiment involves 256 MPI processes spread over four nodes, and the results and inter-node communication pattern are presented in Fig. 4(e)–(f). This example also arises from a realistic case of partitioning an unstructured computational mesh. Every MPI process only sends and receives inter-node messages, where the number of neighbors per process and the message sizes vary considerably. (The maximum number of neighbors per process is 58 whereas the minimum is 0.) The extended max-rate model (4) has been used to estimate the slowest process time usage per node. Per-process estimates are not included to prevent cluttering the plot.

To quantify the accuracy of our model (7)–(9), we calculate a total relative error defined as follows:

$$\text{Total relative error} = \frac{\sum_{i=0}^{P-1} |T_i^{\text{actual}} - T_i|}{\sum_{i=0}^{P-1} T_i^{\text{actual}}}, \tag{11}$$

where T_i^{actual} denotes the measured actual time usage on process i and T_i denotes the per-process model estimate. Using this error definition, we have calculated the modeling error for the experiments shown in Fig. 4 to be 11.5% for the intra-socket experiment, 13.0% for the inter-socket experiment and 12.8% for the inter-node experiment, see Table II.

As shown in Fig. 4(a)–(c), the extended max-rate model under-estimates the per-process time for the intra- and inter-socket scenarios. This is not surprising because it incorrectly models the bandwidth contention. Thus our one-level model (7)–(9) gives better accuracy than the extended max-rate models for intra- and inter-socket traffic. For the third experiment, shown in Fig. 4(e), our model has approximately the same accuracy as the extended max-rate model. This result is expected because two competing MPI processes can already saturate the maximum inter-node bandwidth on this system. A more elaborate model of the bandwidth contention thus does not pay off.

E. Model for Mixing Intra- and Inter-Socket Messages

The two new models (5)–(6) and (7)–(9) can be used to quantify the per-process time usage when all the point-to-point communication happens on one level. However, communications that concurrently take place on different levels can affect each other. We will thus develop a third model to handle a mixture of intra-socket and inter-socket (intra-node) messages. This is important in practice because point-to-point MPI communications on these two levels are particularly subject to the bandwidth contention, which is in essence the contention for the shared memory bandwidth.

The main idea is that the available communication bandwidth, which is to be shared among N competing processes, is neither the pure intra-socket $BW_{\text{MP}}^{\text{on}}(N)$ value nor the pure inter-socket $BW_{\text{MP}}^{\text{off}}(N)$ value. For process i, its achievable bandwidth will depend on the ratio between these two traffic types, more specifically

$$BW_{\text{MP}}^{\text{mix}}(N, \theta_i) = \frac{\theta_i}{N} \cdot BW_{\text{MP}}^{\text{on}}(N) + \frac{1-\theta_i}{N} \cdot BW_{\text{MP}}^{\text{off}}(N),$$
$$\theta_i = \frac{V_i^{\text{on}}}{V_i^{\text{on}} + V_i^{\text{off}}}, \tag{12}$$

where V_i^{on} and V_i^{off} denote, respectively, the total intra- and inter-socket incoming volumes on process i. To model the per-process receiving time usage t_i^{recv}, when intra- and inter-socket traffic is mixed, we start with dividing all the MPI processes into groups based on which socket they reside on. As in the two preceding models, N denotes the number of processes in a group. The challenge now is that we can no longer easily see beforehand the exact order in which the processes will finish receiving their incoming traffic. This is due to a more dynamic competition for the available bandwidth. We will thus adopt a modeling algorithm that is based on yet another staircase strategy consisting of N steps, where in each step we can find out, "on-the-fly," which process will be the next one to finish.

For step $k = 0, 1, \ldots, N-1$:

Find an unfinished process q that gives

$$t_{\text{step}}^{k} = \min_{i} \frac{V_i^{\text{on}} - V_i^{\text{on,recv}} + V_i^{\text{off}} - V_i^{\text{off,recv}}}{BW_{\text{MP}}^{\text{mix}}(N-k, \theta_i)}; \tag{13}$$

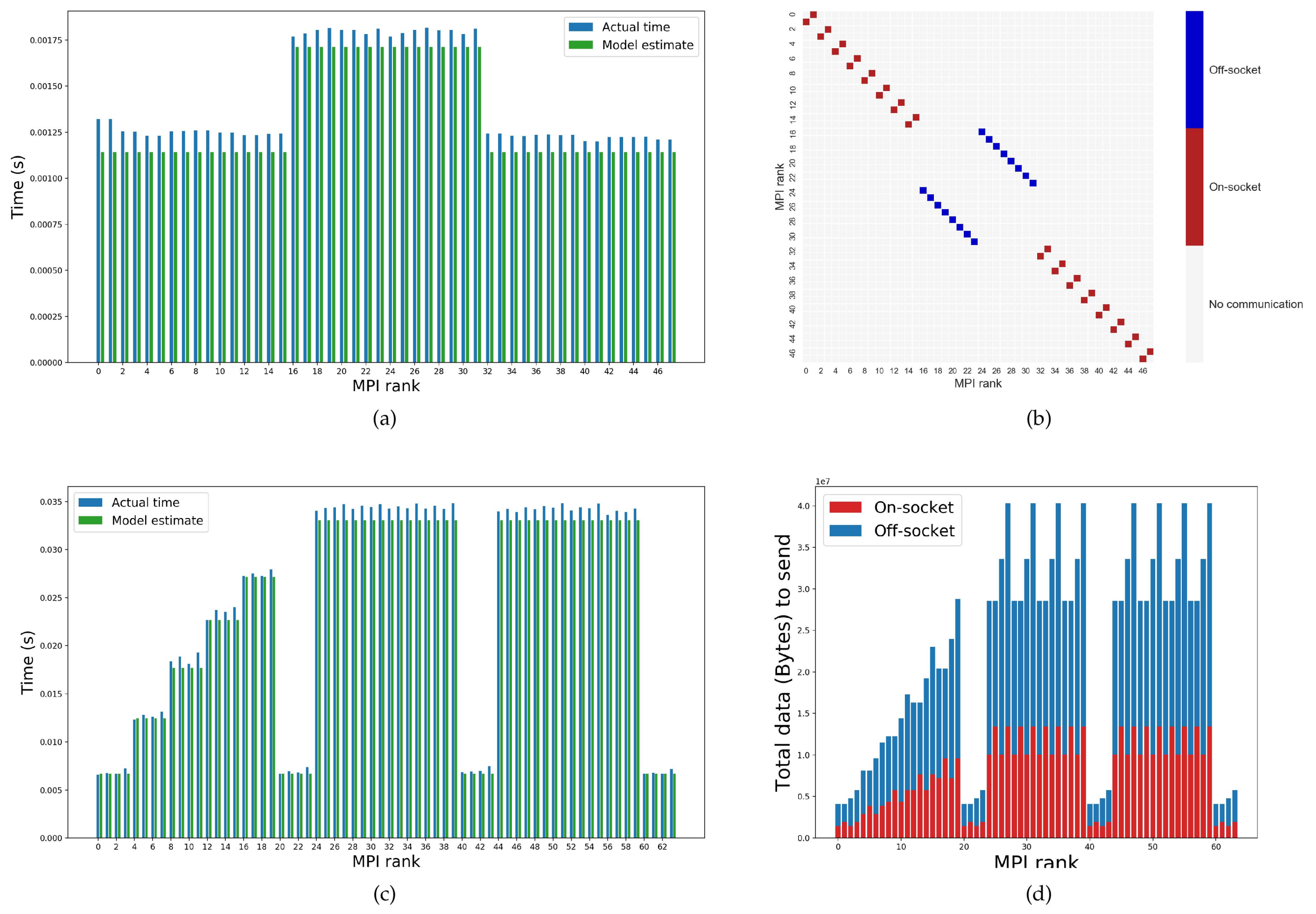


Fig. 5. Plot (a) shows the actual time (blue) and model estimate (green) for an experiment on the dual-socket ThunderX2 machine using 48 processes (evenly spread over two sockets), with the communication pattern shown in plot (b). Plot (c) shows the actual time and model estimate for an experiment on the dual-socket Kunpeng machine using 64 processes (evenly spread over two sockets). In this experiment each process sends and receives three messages of a varying size. The amount of on- and off-socket traffic each process receives is displayed in plot (d).

$$\text{Compute } t_q^{\text{recv}} = \sum_{\ell=0}^{k} t_{\text{step}}^{\ell};\ (\text{process } q \text{ finishes after step } k) \tag{14}$$

$$\text{Compute } T_q = M_q^{\text{in,on}} \cdot \tau^{\text{on}} + M_q^{\text{in,off}} \cdot \tau^{\text{off}} + \max\left(t_q^{\text{recv}}, t_{q,0}^{\text{send}}, t_{q,1}^{\text{send}}, \ldots, t_{q,M_q^{\text{out}}-1}^{\text{send}}\right); \tag{15}$$

For all unfinished processes:

$$V_i^{\text{on,recv}} += t_{\text{step}}^k \cdot \theta_i \cdot BW_{\text{MP}}^{\text{mix}}(N-k, \theta_i); \tag{16}$$

$$V_i^{\text{off,recv}} += t_{\text{step}}^k \cdot (1-\theta_i) \cdot BW_{\text{MP}}^{\text{mix}}(N-k, \theta_i). \tag{17}$$

In the above model, we have used $V_i^{\text{on,recv}}$ and $V_i^{\text{off,recv}}$ to record the accumulated volumes of intra- and inter-socket data received so far on process i. The values of $V_i^{\text{on,recv}}$ and $V_i^{\text{off,recv}}$ are increased during each step using (16)–(17). After each step, a process q will finish, as expressed by (13)–(14). On each process, its individual ratio θ_i remains the same throughout the entire process. A source code that implements the above mixed-level model can be found in the appendix, available in the online supplemental material.

E. Verification examples: We illustrate the mixed intra/inter-socket model (12)–(17) with two simple examples, one using the dual-socket ThunderX2 machine and the other using the dual-socket Kunpeng machine (see Table I in Section II-B). In the first experiment we use in total 48 processes, where 32 processes (16 on each socket) send and receive intra-socket messages and the remaining 16 processes (8 on each socket) send and receive inter-socket messages. The results are presented in Fig. 5(a), and the communication pattern is presented in Fig. 5(b). The plot in Fig. 5(a) includes actual per-process time measurements as blue bars and the staircase model estimate as green bars. Notice that the mixed intra/inter-socket model correctly predicts that the off-socket processes complete after the on-socket processes. Using the error metric defined in (11), the total relative error is found at 6.6% for the staircase estimate.

In the second experiment, we use 64 processes spread over two Kunpeng sockets. The process connectivity is the same as shown in Fig. 4(b), but this time we deliberately “reshuffle” the process mapping (by the OpenMPI option `-map-by socket`) to incur a mixture of intra- and inter-socket communication. The per-process message sizes and types are displayed in Fig. 5(d). The actual time measurements and staircase model estimates are displayed in Fig. 5(c). The resulting total relative error is 3.6%.

TABLE III
EXPERIMENTS WITH REALISTIC COMMUNICATION PATTERNS ON THUNDERX2, KUNPENG, XEON-GOLD AND EPYC ROME. THE RESULTS ARE OBTAINED ON ONE, TWO AND FOUR NODES, USING UP TO 256 PROCESSES ON THUNDERX2, UP TO 512 PROCESSES ON KUNPENG AND EPYC ROME, AND UP TO 208 PROCESSES ON XEON-GOLD. THE TABLE DISPLAYS THE TOTAL NUMBER OF MESSAGES ($\sum$M), MAX PER-PROCESS MESSAGES, TOTAL COMMUNICATION VOLUME (IN MBS), THE ON-SOCKET (INTRA-SOCKET), OFF-SOCKET (INTER-SOCKET) AND INTER-NODE PERCENTAGES OF THE TOTAL VOLUME, AND THE TOTAL RELATIVE PREDICTION ERROR

ThunderX2								Kunpeng							
#procs	$\sum$M	maxM	vol	on	off	inter	error	#procs	$\sum$M	maxM	vol	on	off	inter	error
32	154	9	17	34%	66%	0%	0.142	32	154	9	17	34%	66%	0%	0.097
64	374	11	30	36%	64%	0%	0.157	64	374	11	30	36%	64%	0%	0.085
128	1180	22	68	27%	70%	3%	0.158	128	1180	22	68	27%	73%	0%	0.115
256	4022	33	248	35%	62%	3%	0.189	256	4022	33	248	36%	63%	1%	0.075
								512	14650	52	403	43%	56%	1%	0.110
Xeon-Gold								**Epyc Rome**							
#procs	$\sum$M	maxM	vol	on	off	inter	error	#procs	$\sum$M	maxM	vol	on	off	inter	error
26	132	10	13	24%	76%	0%	0.152	64	1394	46	153	37%	63%	0%	0.093
52	304	12	22	37%	63%	0%	0.149	128	2944	91	280	38%	62%	0%	0.143
104	760	14	41	29%	69%	2%	0.122	256	5118	104	544	36%	61%	3%	0.133
208	2446	24	123	25%	73%	2%	0.156	512	13552	104	1116	35%	57%	8%	0.141

IV. REALISTIC TESTS

So far, we have mostly used synthetic experiments, with the only exception being those shown in Fig. 4(c)–(f). In this section, we will apply the new performance models to realistic communication patterns.

A. *Mixing Intra-Node and Inter-Node Messages*

The most general point-to-point MPI communication can simultaneously take place at all the levels: intra-socket, inter-socket and inter-node. Our approach is to use the one-level model (7)–(9) for estimating the inter-node cost per process as $T_i^{\text{inter-node}}$, whereas the mixed-level model (12)–(17) is used for estimating the mixed intra/inter-socket cost as $T_i^{\text{intra-node}}$. The total time per process is given by

$$T_i^{\text{total}} = T_i^{\text{inter-node}} + T_i^{\text{intra-node}}. \tag{18}$$

The above model of total time cost assumes that the intra-node and inter-node communication tasks cannot proceed simultaneously, whereas the intra-socket and inter-socket messages can compete for the same memory bandwidth (while dynamically affecting each other). We remark that summing up the inter-node costs with the intra-node costs is the same approach as adopted in [6].

B. *Realistic Communication Patterns*

For realistic communication patterns, we have adopted some examples of partitioning of realistic 3D unstructured meshes that have been used for reservoir simulations [10], [11], [12]. This is in connection with an MPI parallelization of a reservoir simulator, where Algorithm 2 needs to be repeatedly executed when numerically solving the involved partial differential equations. We refer the reader to [12] for the details. Given a decomposition of the mesh, based on a specified number of MPI processes, we can create the corresponding communication pattern of the irregular point-to-point MPI communication operations. Each MPI message is classified as on-socket, off-socket or inter-node.

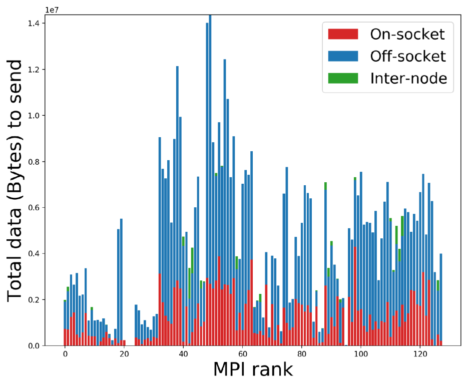
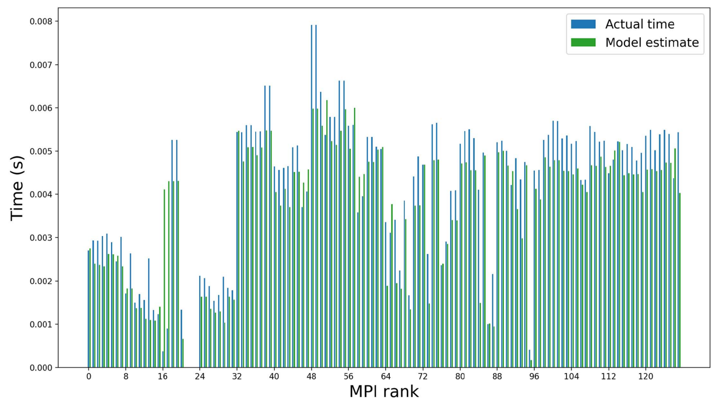


Fig. 6. One example of realistic per-process communication volumes when using 128 MPI processes spread over two compute nodes (with four sockets in total).

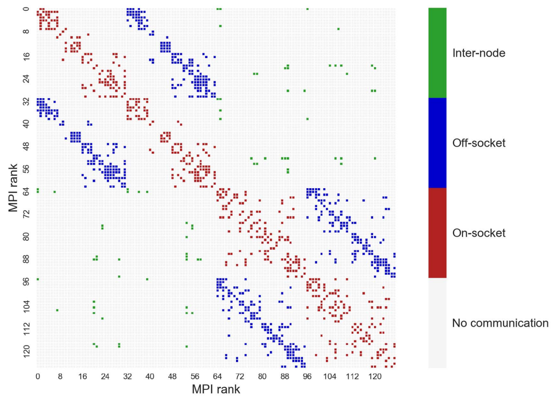
C. *Small-Scale Experiments*

To test the performance model (18) on realistic communication patterns, we compare the per-process execution times of Algorithm 2 with the model estimates when running different numbers of MPI processes on four computing platforms: ThunderX2, Kunpeng, Xeon-Gold and Epyc Rome (see Table I in Section II-B), using up to four compute nodes. One realistic example of per-process communication volumes can be seen in Fig. 6.

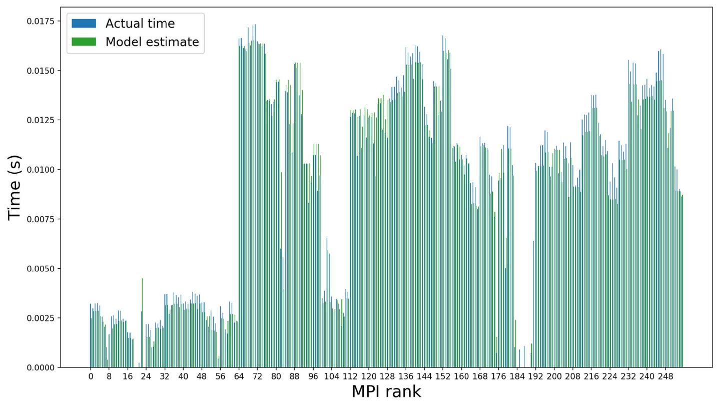
Table III shows that the performance model (18) achieves good prediction results. The total relative error ranges from 7.5% to 18.9% on ThunderX2 and Kunpeng, from 12.2% to 15.6% on Xeon-Gold, and from 9.3% to 14.3% on Epyc Rome. A more detailed comparison for two ThunderX2 and two Kunpeng nodes is presented in Fig. 7. Here, the per-process execution time and model estimate are displayed in bar plots. In Fig. 7(a), execution times and model estimates for 128 processes on two ThunderX2 nodes are displayed, while the plot in Fig. 7(b) displays the timing results and model estimates for 256 processes on two Kunpeng nodes. We can observe good correspondence between the model estimates and execution times for most cases.



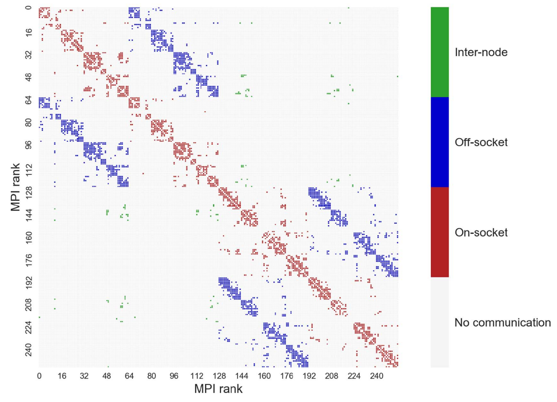
(a)



(b)



(c)



(d)

Fig. 7. The per-process execution times (blue bars) and model estimates (green bars) for two realistic communication patterns on, respectively, two ThunderX2 nodes (a)–(b) and two Kunpeng nodes (c)–(d).

D. Large-Scale Experiments

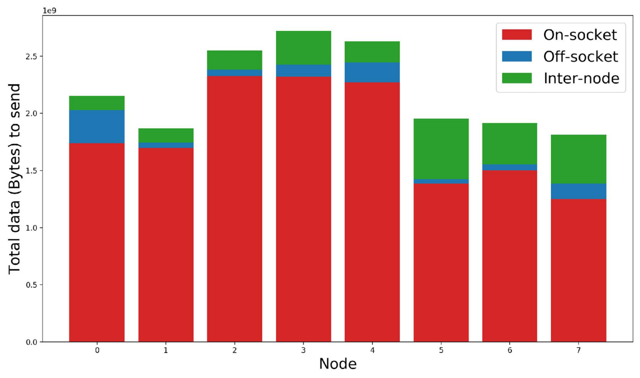
We have also tested our model on the Betzy supercomputer [13], which is a BullSequana XH2000 system with in toal 1344 compute nodes. Each compute node has two 64-core AMD Epyc Rome CPUs for a total of 172,032 CPU cores. The system uses an Infiniband HDR 100 network connected in a Dragonfly+ topology [14], which consists of 14 groups each with 96 nodes. The Dragonfly+ topology improves scalability over the original Dragonfly, while maintaining the cost benefits when compared with the Fat-tree topology [15]. The Dragonfly+ topology also provides full bisection bandwidth for intra-group communications, but inter-group communications are oversubscribed and require the use of non-minimal routing to achieve good performance. This means that the Betzy system will see excellent inter-node communication for MPI processes running inside a single group, but performance may suffer with regards to both bandwidth and latency when the MPI processes are spread across two or more groups, depending on the overall network load.

The prediction accuracy of our model (18) has been tested on Betzy using communication patterns that include a larger proportion of inter-node messages arising from partitioning a large reservoir mesh that is ten times larger than that used for the experiments on ThunderX2, Kunpeng and Xeon-Gold in Section IV-C. Fig. 8 shows a breakdown of the per-node communication volumes for this large case when being partitioned into 1024 and 8192 subdomains, using 8 and 64 compute nodes on Betzy, respectively.

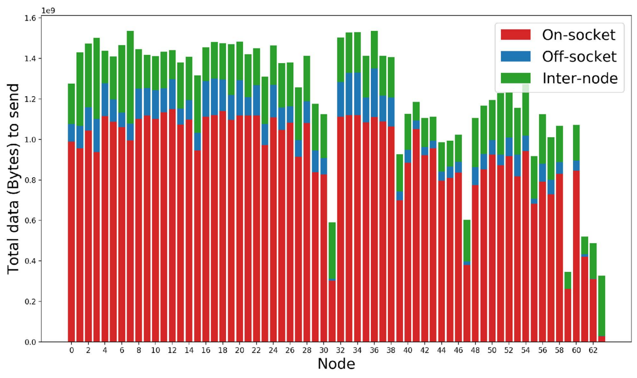
From Fig. 8 we can observe that the communication patterns attained by partitioning the large mesh result in a majority of on-socket communication. However, we also notice that there is a significant amount of off-socket and inter-node communication. The model prediction accuracy for our large-scale Epyc Rome experiments is presented in Table IV. In this table, we include results attained on 4 to 64 nodes. In the scenario of 64 nodes, the MPI processes are distributed (unevenly) across two Dragonfly+ groups in order to enforce non-uniform network connectivity and test performance accuracy when both inter- and intra-group communications are present. The total relative prediction error displayed in Table IV ranges from 12.2% to 20.9%, with the error being 16.2% for 8192 processes on 64 nodes.

V. Related Work

For a long time, the de facto standard for analyzing the communication performance of parallel systems was the postal model [1], [2]. This two-parameter model has the benefit of being



(a)



(b)

Fig. 8. Per-node communication volumes when partitioning a realistic reservoir mesh into 1024 and 8192 subdomains on 8 and 64 AMD Epyc Rome nodes.

TABLE IV
STUDY OF PER-PROCESS MPI OVERHEAD WITH REALISTIC COMMUNICATION PATTERNS ON A CLUSTER OF AMD EPYC ROME NODES. THE ROWS SHOW, RESPECTIVELY, THE NUMBER OF MPI PROCESSES (NODES), TOTAL NUMBER OF MESSAGES, MAXIMUM PER-PROCESS NUMBER OF MESSAGES, TOTAL COMMUNICATION VOLUME, MAXIMUM PER-PROCESS COMMUNICATION VOLUME, MINIMUM PER-PROCESS VOLUME, OVERALL INTRA-SOCKET COMMUNICATION VOLUME PERCENTAGE, INTER-SOCKET VOLUME PERCENTAGE, INTER-NODE VOLUME PERCENTAGE, AND THE TOTAL RELATIVE PREDICTION ERROR FOR THE STAIRCASE ESTIMATE

#procs (nodes)	512(4)	640(5)	768(6)	896(7)	1024(8)	2048(16)	4096(32)	8192(64)
#messages	13552	22598	29866	41844	50756	268036	1050652	2744632
Max messages	104	125	164	182	210	375	733	1235
Total vol (MB)	1116	1353	1524	1923	2098	4801	7541	9382
Max vol (MB)	6.68	8.15	7.44	6.62	5.96	4.15	2.82	1.86
Min vol (KB)	0.88	0.59	1.17	5.27	0.59	0.29	0.59	0.29
Intra-socket	83.9%	71.6%	72.4%	74.9%	82.3%	84.3%	81.9%	74.8%
Inter-socket	8.9%	7.7%	10.8%	7.0%	5.1%	3.9%	4.5%	8.0%
Inter-node	7.2%	20.7%	16.8%	18.1%	12.6%	11.8%	13.6%	17.2%
Prediction error	0.145	0.209	0.181	0.158	0.178	0.140	0.122	0.162

very simple, but its performance estimates are only accurate for fix sized, single messages on massively parallel processing (MPP) systems. These traditional MPP systems were homogeneous and there was only inter-node communication (i.e., no intra-node communication). The limitation of single messages was addressed in the more recent max-rate model [3], which extends the postal model to support N concurrent messages from N send-receive process pairs. Improvements of the max-rate model were introduced in [16] to clearly distinguish between inter-node, intra-node and intra-socket messages and to estimate network contention. Nevertheless, the max-rate model and its successor still lack the support for variable sized messages, plus the weakness of adopting an over-simplified "roofline" model for quantifying the aggregate bandwidth that is available to multiple sender-receiver pairs. In comparison, our work in the present paper adopts accurate aggregate bandwidth values that are measured by a simple bi-directional, multi-pair micro-benchmark, which was extended from the `osu_bibw` benchmark of MVAPICH [4]. The main difference between our work and the max-rate models [3], [16] is the adoption of a *staircase* modeling to quantify the per-process cost for the general cases where different MPI processes can have different numbers of neighbors and the messages are of different sizes. Apart from modeling point-to-point communications on a single interconnection level, our models can specifically handle the dynamic competition between concurrent intra-socket and inter-socket messages.

Apart from the postal model and the max-rate models, another body of work on communication modeling contains the LogP family of models. This originated in the LogP model which was first proposed by Culler et al. in [2]. The LogP model adds communication overhead as an additional parameter to better estimemate the cost of sending and receiving messages, but is still constrained to fix sized, single messages and limited in accuracy for long messages. The latter was improved in the LogGP model [17] which extends the LogP model with support for linear modeling of long messages. Several other refinements also exist, including LogGPC [18] which adds parameters for network contention, LogGPS [19] which estimates synchronisation costs in the communications library, and LogGPO [20] which characterizes the overlap between communication and computation. A comprehensive survey of the different models was provided in [21]. The main drawback of the LogP-family models is the adoption of hard-to-estimate parameters, such as network contention and synchronisation costs, in addition to the lack of support for variable sized messages and heterogeneous levels of latency and bandwidth. In comparison, instead of detailing the different contributing terms to the overall latency/overhead, which is the main modeling principle adopted by the LogP family, we let a micro-benchmark quantify the overall latency/overhead and focus instead on modeling the dynamic competition between variable sized, concurrent messages.

The particular modeling challenges due to the wide use of shared-memory nodes in modern compute clusters, together with the resulting concurrent messages, were investigated in the τ-Lop model [22] and its extension [23]. Our work follows the same notion of *concurrent transfers* from [22] and the reasoning about the resulting contention among multiple MPI processes for the same bandwidth. However, our "staircase" modeling strategy for bandwidth contention is completely different that used in [22].

VI. Conclusion

The basic assumption for the work presented in the current paper is that concurrent MPI messages that are transmitted over the same connection of a communication network should *fairly* share the aggregate bandwidth available. Moreover, when the messages are of different sizes, the principle of fairness will lead to that the shortest message should complete first, followed by the second shortest one, and so on. Such a dynamically changing situation of competition will, at the same time, lead to a dynamic change of the available bandwidth, which may not follow a simple profile (3) as suggested by the max-rate model. This has inspired the "staircase" modeling strategy that is heavily used in our new performance models.

In reality, the actual completion sequence of the different messages may indeed not follow the fairness principle. Various stochastic features will lead to different sequences even when an MPI program is repeated several times on the same parallel system while using the same configuration. We consider it impractical to properly model such stochasticity. Instead, we adopt an "idealized" expectation based on fairness. Despite this simplification, numerical experiments reported in Sections III and IV have shown a consistently good agreement between the model predictions and the actual time measurements. In particular, a realistic case that involves 8192 MPI processes and 2.7 million concurrent messages (see Table IV) has achieved an average overall accuracy of 83.8%.

Overall, our "staircase" modeling strategy is manageably simple. The models from Sections III-D and III-E can be efficiently implemented such as shown in the appendix, available in the online supplemental material. As preparation work before using our new models, the bi-directional, multi-pair micro-benchmark (Algorithm 1) needs to be executed for different MPI process counts and for three situations: within a CPU socket, between a pair of CPU sockets (with shared memory), and between a pair of nodes in a cluster. The measured values of τ and BW_{MP} can be tabulated once and for all. To save the preparation work, the τ and BW_{MP} values for the short and eager protocols can be skipped, if the message sizes of interest are known to lie in the rendezvous regime.

In this paper, the possible costs that are associated with packing and unpacking MPI messages are not included. Such extra costs are necessary if the outgoing messages are composed of data values that lie non-contiguously in the sender's memory, or the values of the incoming messages need to be placed in designated, non-contiguous locations in the receiver's memory. There exist models, such as $\log_n$P or $\log_3$P [24], that include *regularly strided* memory accesses, which are the simplest operations for packing and unpacking messages. As future work, we will extend our models to cover the packing and unpacking costs, and we will address these from the angle of memory bandwidth contention. Specifically, we will consider the situation where a preceding computation produces the values before they are collected from *irregularly* non-contiguous memory locations and packed as outgoing messages. Moreover, the different MPI processes on a same node may carry out the "packing step" at different paces (e.g., some processes have less data to pack and can thus initiate the MPI commands earlier), such that the message transfer and message packing by different MPI processes may compete for the same memory bandwidth. As another topic of future work, we will consider modeling overlaps between computation (including message packing and unpacking) and point-to-point MPI communication. Contention for the same memory bandwidth, see e.g. [25], [26], will also be central in this regard.

One of the main benefits of using our "staircase" modeling approach is the capability of pinpointing per-process time usage, thanks to an elaborate modeling of the bandwidth contention between messages of various sizes. Realistic experiments have seen actual time usages greatly varying from process to process, thus the fastest finishing MPI processes may have considerable idle time. We plan to incorporate such modeling into future mesh partitioning strategies, with the goal of producing better partitioning results where the fastest finishing processes do not have to wait long for the slowest processes to complete.

References

[1] R. W. Hockney and C. R. Jesshope, *Parallel Computers Two: Architecture, Programming and Algorithms*, 2nd ed., Bristol, U.K.: IOP Publishing Ltd., 1988.

[2] A. Bar-Noy and S. Kipnis, "Designing broadcasting algorithms in the postal model for message-passing systems," in *Proc. 4th Annu. ACM Symp. Parallel Algorithms Architectures*, 1992, pp. 13–22.

[3] W. Gropp, L. N. Olson, and P. Samfass, "Modeling MPI communication performance on SMP nodes: Is it time to retire the ping pong test," in *Proc. 23rd Eur. MPI Users' Group Meeting*, 2016, pp. 41–50.

[4] "MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE," 2022. [Online]. Available: https://mvapich.cse.ohio-state.edu/benchmarks/

[5] R. W. Hockney, "The communication challenge for MPP: Intel paragon and meiko CS-2," *Parallel Comput.*, vol. 20, no. 3, pp. 389–398, 1994.

[6] A. Bienz, W. D. Gropp, and L. N. Olson, "Reducing communication in algebraic multigrid with multi-step node aware communication," *Int. J. High Perform. Comput. Appl.*, vol. 34, no. 5, pp. 547–561, 2020.

[7] P. Bastian et al., "A generic grid interface for parallel and adaptive scientific computing. part II: Implementation and tests in DUNE," *Computing*, vol. 82, no. 2/3, pp. 121–138, 2008.

[8] O. Sander, *DUNE — The Distributed and Unified Numerics Environment*, Cham, Switzerland: Springer, 2020.

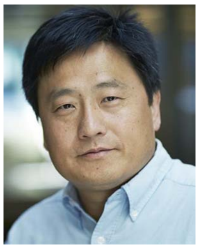
[9] "DUNE Numerics," 2021, Accessed: May 2022. [Online]. Available: https://dune-project.org

[10] "OPEN POROUS MEDIA," 2022, Accessed: May 2022. [Online]. Available: https://opm-project.org
[11] A. F. Rasmussen et al., "The open porous media flow reservoir simulator," *Comput. Math. Appl.*, vol. 81, pp. 159–185, 2021.
[12] A. Thune, X. Cai, and A. B. Rustad, "On the impact of heterogeneity-aware mesh partitioning and non-contributing computation removal on parallel reservoir simulations," *J. Math. Ind.*, vol. 11, 2021, Art. no. 12.
[13] "Betzy–The most powerful supercomputer in Norway." [Online]. Available: https://documentation.sigma2.no/hpc_machines/betzy.html
[14] A. Shpiner, Z. Haramaty, S. Eliad, V. Zdornov, B. Gafni, and E. Zahavi, "Dragonfly: Low cost topology for scaling datacenters," in *Proc. IEEE 3rd Int. Workshop High- Perform. Interconnection Netw. Exascale Big-Data Era* , 2017, pp. 1–8.
[15] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *Proc. Int. Symp. Comput. Architecture*, 2008, pp. 77–88.
[16] A. Bienz, W. D. Gropp, and L. N. Olson, "Improving performance models for irregular point-to-point communication," in *Proc. 25th Eur. MPI Users' Group Meeting*, 2018, pp. 1–8.
[17] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "LogGP: Incorporating long messages into the LogP model—one step closer towards a realistic model for parallel computation," in *Proc. seventh Annu. ACM Symp. Parallel Algorithms Architectures*, 1995, pp. 95–105.
[18] C. A. Moritz and M. I. Frank, "LoGPC: Modeling network contention in message-passing programs," in *Proc. ACM SIGMETRICS Joint Int. Conf. Meas. Model. Comput. Syst.*, 1998, pp. 254–263.
[19] F. Ino, N. Fujimoto, and K. Hagihara, "LogGPS: A parallel computational model for synchronization analysis," *SIGPLAN Not.*, vol. 36, no. 7, pp. 133–142, Jun. 2001. [Online]. Available: https://doi.org/10.1145/568014.379592
[20] W. Chen, J. Zhai, J. Zhang, and W. Zheng, "LogGPO: An accurate communication model for performance prediction of MPI programs," *Sci. China Ser. F: Inf. Sci.*, vol. 52, no. 10, pp. 1785–1791, 2009.
[21] J.-A. Rico-Gallego, J.-C. Díaz-Martín, R. R. Manumachu, and A. L. Lastovetsky, "A survey of communication performance models for high-performance computing," *ACM Comput. Surv.*, vol. 51, no. 6, 2019, Art. no. 126.
[22] J.-A. Rico-Gallego and J.-C. Díaz-Martín, "-Lop: Modeling performance of shared memory MPI," *Parallel Comput.*, vol. 46, pp. 14–31, 2015.
[23] J.-A. Rico-Gallego, J.-C. Díaz-Martín, and A. L. Lastovetsky, "Extending -Lop to model concurrent MPI communications in multicore clusters," *Future Gener. Comput. Syst.*, vol. 61, pp. 66–82, 2016.
[24] K. W. Cameron, R. Ge, and X.-H. Sun, "P and P: Accurate analytical models of point-to-point communication in distributed systems," *IEEE Trans. Comput.*, vol. 56, no. 3, pp. 314–327, Mar. 2007.
[25] J. Langguth, X. Cai, and M. Sourouri, "Memory bandwidth contention: Communication vs computation tradeoffs in supercomputers with multicore architectures," in *Proc. IEEE 24th Int. Conf. Parallel Distrib. Syst.*, 2018, pp. 497–506.
[26] A. Denis, E. Jeannot, and P. Swartvagher, "Interferences between communications and computations in distributed HPC systems," in *Proc. 50th Int. Conf. Parallel Process.*, 2021, pp. 1–11.

Andreas Thune received the MSc degree from the Institute of Mathematics, University of Oslo, in 2017. He is currently working toward the PhD degree with the University of Oslo and Simula Research Laboratory. His research topic is high performance computing in the domain of reservoir simulation.

Sven-Arne Reinemo (Senior Member, IEEE) received the cand.scient. and dr.scient. degrees in computer science from the University of Oslo, Norway, in 2000 and 2007, respectively. He is the research director with Simula Metropolitan Center for Digital Engineering. His current research interests are interconnection networks, high performance computing, communication systems and cyber security.

Tor Skeie is a professor with the University of Oslo and Simula Research Laboratory; his research has mainly contributed to the High-Performance Computing field (HPC). Herein he has focused on effective routing, fault tolerance, congestion control, quality of service, reservoir simulations, edge and cloud computing. He is also a researcher with the Industrial Ethernet and wireless networking areas. Several of his research results have been published in the most respected IEEE transactions and magazines. More details about Prof. Skeie are available with http://simula.no/people/tskeie.



Xing Cai received the PhD degree in scientific computing from the University of Oslo, in 1998, and was appointed to the position of associate professor with the University of Oslo, in 1999, later promoted to full professorship in 2008. He joined Simula with its very beginning in 2001, taking an 80% leave from his university position. His research interests include parallel programming and high-performance scientific computing on multi-core CPUs and GPUs, numerical methods for solving PDEs, and generic PDE software.