Model-Based Estimation of the Communication Cost of Hybrid Data-Parallel Applications on Heterogeneous Clusters

Juan-Antonio Rico-Gallego, Alexey L. Lastovetsky, Member, *IEEE*, and Juan-Carlos Díaz-Martín

Abstract—Heterogeneous systems composed of CPUs and accelerators sharing communication channels of different performance are getting mainstream in HPC but, at the same time, they show a complexity that makes it difficult to optimize the deployment of a data parallel application. Recent analytical tools such as Functional Performance Models, combined with advanced partitioning algorithms, manage to achieve a balanced configuration by distributing the workload unevenly, according to the performance of the different processing units. Unfortunately, such uneven distribution of the computation load leads to communication unbalances that, very often, render worthless the previous workload balancing efforts. Finding the optimal communication scheme without expensive testing on the executing platform requires an analytical approach to the estimation of the communication cost of different configurations of the application. With this goal in mind, we propose and discuss an extension of the τ–Lop communication performance model to cover heterogeneous architectures. In order to provide a quantitative assessment of this extended model, we conduct experiments with two representative computational kernels, the SUMMA algorithm and the 2D wave equation solver. The τ–Lop predictions are compared against the HLogGP model and the observed costs for a variety of configurations, hardware resources and problem sizes.

Index Terms—Communication Performance Modeling, Hybrid Algorithms, Performance Analysis, Functional Performance Models, Heterogeneous Platforms.

1 Motivation and goals

The efficient increase of performance of modern HPC systems, measured in terms of both cost and energy consumption, is the main cause for high performance computing becoming heterogeneous. Systems composed of nodes with multi-core processors and accelerators, and communicating through channels of different capacity, are getting mainstream. Data parallel applications running on these platforms, known as *hybrid applications*, distribute data between a set of processes that run on processors of different types and hence with distinct performance. We know this set as a *process layout*. It is described in a file with the number, type and mapping of the processes onto the platform processors. Fig. 1 shows an example.

A *hybrid application* typically runs on top of a set of one or more *kernels*. A kernel is a recurrent representative piece of code, such as matrix multiplication or Fourier transform, which usually progresses by repeating two stages: computation and communication. To achieve optimal performance, a hybrid application unevenly distributes the kernel workload between its processes, thus avoiding faster processes to wait for slower processes at synchronization or communication points. Though current research on computation performance models has disregarded communication costs, the fact is that they are significant and such non-uniform distribution has an influence on them. The reason is twofold. First, non-uniform distribution introduces high traffic imbalances, because different processes communicate different volumes of data. Second, the mapping of processes to the processors of the platform determines their communication channels. These channels have different features that significantly impact on the global communication cost. The pair composed by the distribution of the kernel workload among the processes, and by the mapping of these processes on the platform is known as a *configuration*. The communication cost of a hybrid application will depend on the chosen configuration. The point is that the configuration

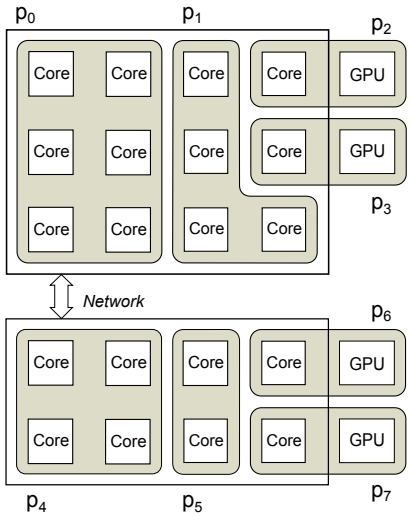


Fig. 1: Process layout of eight processes $p = \{p_0, p_1, \ldots, p_7\}$ in a machine of two nodes of different types with GPUs. GPU processes use a dedicated core for communication.

Juan-Antonio Rico-Gallego and Juan-Carlos Díaz-Martín are with the University of Extremadura, Avd. Universidad s/n, 10003, Cáceres, Spain. E-mail: {jarico,juancarl}@unex.es
Alexey L. Lastovetsky is with the University College Dublin, Belfield, Dublin 4, Ireland.E-mail: alexey.lastovetsky@ucd.ie

achieving optimal computation performance may suffer an adverse communication pattern that ruins the previous load balancing efforts. It is this problem that motivates this work.

Especially for large multi-kernel applications, the estimation of the cost of communication is usually carried out through a battery of thorough tests of a shortened version of the target application (for instance, individual kernels) on a subset of the target platform. This practice has two main drawbacks: the use of valuable computational resources for testing, and the difficulty (often, the impossibility) of correctly extrapolating estimations from such twofold simplification. This scenario calls for a fully analytic approach, shown in Fig. 2, that adds a *communication performance model* to the *computation performance model* used to determine load balanced configurations. Needless to say, only this standpoint avoids the long, difficult and expensive experimental testing, allowing quick comparisons of multiple settings to, eventually, find out that of the optimal overall cost.

Indeed, the cornerstone of this approach is an appropriate model. Our proposal is an extension of τ–Lop, a communication performance model originally designed to predict the cost of collective operations in homogeneous multi-core clusters. The main contributions of this paper are:

C1. An extension of the τ–Lop model for estimation of the communication cost of data-parallel application on hybrid heterogeneous platforms.
C2. Experimental validation of the accuracy of the extended model using two data parallel kernels with opposite features, the SUMMA matrix multiplication, with a high communication to computation ratio, and a finite difference solver of the 2D wave equation, much less demanding in terms of communication.
C3. An automatic tool for solving the formal model expressions named τ–Lop Library. Its description and examples of the use are available as supplementary material.

Exhibiting a remarkable economy of parameters, the model predicts the communication cost with good accuracy, robustness and scalability (within the scale of the experimental platform). In our view, there is no a priori reason that could hinder the extrapolation of the experimental results to other applications. The principles of typical data parallel applications such as FFT or molecular dynamics are likewise anchored to a 2D or 3D matrix partitioned among P processes.

The rest of the paper is structured as follows. Section 2 reviews the field of performance optimization in heterogeneous platforms, focusing on computational load balancing. Section 3 introduces the τ–Lop model and extends it in order to cover heterogeneous platforms. Section 4 describes the kernels used to evaluate τ–Lop. Section 5 presents the evaluation method and its results, and Section 6 concludes.

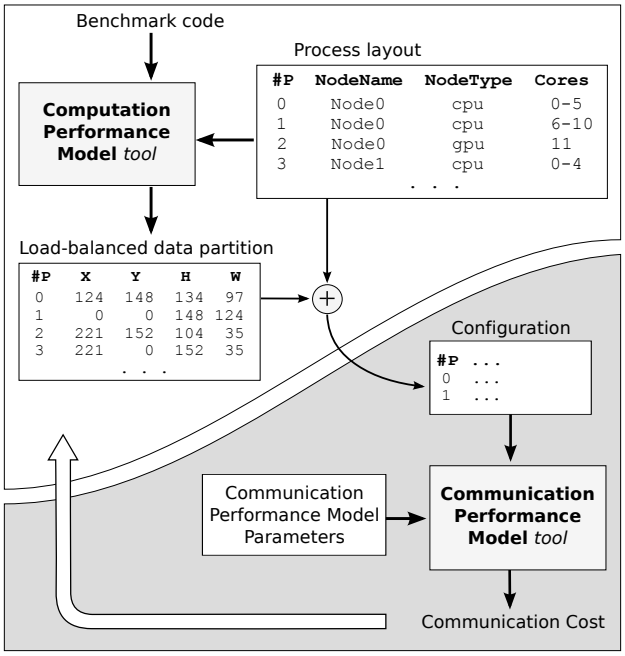


Fig. 2: Fully analytical approach to performance modeling of hybrid data-parallel applications. The partition table describes the coordinates (X and Y) and shape (Width and Height) of the per-process data rectangle. The configuration is the union of the partition and layout tables.

2 Related Work

Two main approaches face the problem of optimizing the performance of applications on heterogeneous platforms. The first characterizes the application as a graph and applies techniques of *graph partitioning*. Though not directly connected with the paper, it is briefly covered here for the sake of completeness and bringing a broader perspective. The second, born in the very field of heterogeneous computing, characterizes the application processes with a *computation performance model* and distributes the computation among them according to their performance.

Graph partitioning is a technique extensively studied. The application is modeled as a graph. Each vertex represents a computational task, and edges represent the communication between the vertices. The graph is partitioned into subsets, and each subset is assigned to a process. The partitioning objective is to minimize the number of edges connecting the subsets, hence minimizing the communication. Methods to accomplish the partition problem are generally classified in *combinational* [1, 2], *spectral* [3, 4], *multilevel* [5, 6] and *geometric* [7, 8], with a large number of algorithms in the literature. This extensive amount of work contrasts, to the best of our knowledge, with its actual use in real-life applications. There exist several tools implementing graph partition algorithms such as METIS [9], JOSTLE [10] and SCOTCH [11]. In addition to the graph partitioning problem, there is an issue of mapping the resultant graph to the underlying non-homogeneous network. Pellegrini discusses different methods for solving this problem in [12]. The matching of the communication pattern of an application, represented by a partitioned graph, to the underlying hierarchical network, represented as a tree, is accomplished by the Jeannot et al. [13] *TreeMatch* algorithm.

The second, more recent approach, deals with determining a balanced distribution of the kernel workload among a pre-specified set of processes on a heterogeneous platform

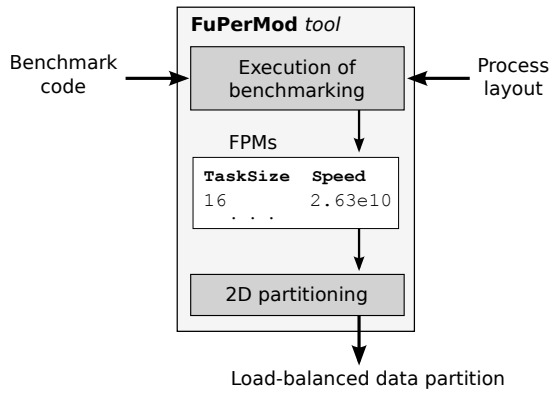


Fig. 3: Outline of the FuPerMod-based tool for 2D data partitioning.

(upper part of Fig. 2) ignoring, at its current state of development, communication costs. As pointed out, the automatic determination of the communication cost of such balanced workload is the focus of this work (lower part of Fig. 2).

The computational load balancing of a hybrid application has been well settled in the literature. Its relevant topics, notwithstanding, need to be recalled here in some detail. Formulated as a data partitioning problem [14], it departs from n independent *computational units* of equal size. A computational unit is kernel dependent. In a matrix multiplication, for instance, the computational unit could be the calculation of an element or, more commonly, a small subregion of the resultant matrix. The goal is to distribute these $\mathcal{N}$ computational units among the set $p = \{p_0, p_1, \ldots, p_{P-1}\}$ of P ($P < \mathcal{N}$) processes in the platform (as those of Fig. 1), in a way that the workload is best balanced. The processes are characterized by the set $s = \{s_0, s_1, \ldots, s_{P-1}\}$, where s_i is the *speed* of p_i, the number of computational units p_i performs per time unit. Such modeling of the performance of a process is referred as *Constant Performance Model*. Lets suppose $n = \{n_0, n_1, \ldots, n_{P-1}\}$, with n_i the number of computational units assigned to p_i, so $\mathcal{N} = \sum_{i=0}^{P-1} n_i$. Each process p_i has an execution time $t_i = \frac{n_i}{s_i}$. The execution time of the application as a whole is given by the slower process as $\max_{i=0}^{P-1} t_i$. An optimal workload distribution minimizes this value. Beaumont et al [15] develops an algorithm for achieving optimal load balancing using a *Constant Performance Model* and applies it to the matrix multiplication algorithm in the ScaLAPACK library [16].

A challenging point is to determine s_i. It depends on so many factors of the platform that representing it as a mere constant results simplistic and inaccurate. For instance, a process can become much slower if its assigned data exceeds the cache size. A *Functional Performance Model* [17] represents the speed of a process as a function of its *task size*, a set of parameters characterizing the amount of its assigned data. As an example, the task size of the matrix multiplication of two square matrices of size $x \times x$ is characterized by x. It defines the amount of data stored as $3 \times x^2$ and the number of operations to do as $(x + (x-1)) \times x^2 \approx 2\,x^3$. The speed of p_i is now a real function of the task size $s_i(x)$.

An algorithm achieving optimal load balancing $\frac{n_i}{s_i(x)} \approx const$ is proposed in [17]. FuPerMod [18, 19] is a software tool that, using this algorithm, implements the whole procedure of workload partition and distribution for a set of processes running on a heterogeneous platform. The FPM of p_i is a table of real values representing s_i in executing a particular kernel for different task sizes. Fig. 3 shows the FuPerMod functionality. First, FuPerMod generates the per-process FPM executing a benchmark code provided by the user. The benchmark code has to be as similar to the kernel code as possible. The user also provides a text file describing the layout of the processes, that is, the execution node and its processor cores or GPU. Next, following with the matrix multiplication example, a 2D partitioning of the matrices is generated from all the FPMs [20, 21, 22] assigning a rectangle of data in the matrix to each process.

Turning to the communication issue, a formal model can be used to estimate the cost of the communications of a hybrid application. To the best of our knowledge, only a limited number of analytic models have been developed to predict the communication cost on heterogeneous platforms. HLogGP [23] is a model based on LogGP [24] that takes into account both the processor and network heterogeneity. The scalar parameters of the homogeneous LogGP model are expanded to represent the values of the P processors as vectors of P components, and the parameters involving links of a pair of processors are expanded to matrices of $P \times P$ values. HLogGP provides a methodology to measure the parameters based on simple micro-benchmarks. The model estimations are validated for a single master/worker application in a small cluster. LMO [25] is another model designed for heterogeneous platforms. It is based on the Hockney model [26], a presumed less accurate model than LogGP, and assumes an Ethernet network without contention. Nevertheless, it has interesting features. LMO defines the cost of a point-to-point message through a set of parameters representing the fixed and variable costs related to the specific processor and the cost derived from the network. It predicts the communication cost of point-to-point, one-to-many (scatter and gather) and broadcasting operations in a switched Ethernet heterogeneous cluster. A disadvantage of these models is the amount of parameters to be measured, which tends to P^2 in a platform with P processors. Despite their interest, they have not proven their accuracy and utility in modeling real applications.

Ogata et al. [27] propose a model for optimizing an FFT library on a GPU/CPU heterogeneous platform. It divides the FFT computation into subsets and predicts the execution time of each step for a particular GPU/CPU combination. The model is interesting because it considers not just computing time, but also the cost of data transfers between CPU and GPU, estimated by a simple linear communication performance model. Chan et al. [28] propose a performance model that predicts execution times of iterative mesh-based applications running on heterogeneous multi-core clusters. The execution time for a process includes the communication time. Both jobs share the feature that their modeling domain is specific, reduced to a particular case, in contrast to τ-Lop, LMO and HLogGP, which take a general approach.

3 The τ–Lop Model and its Extension for Heterogeneous Communication Modeling

τ–Lop [29, 30] is a middleware parameterized communication performance model aimed to represent and predict the cost of parallel algorithms in multi-core clusters. It represents a point-to-point message *transmission* between two processes as a sequence of *transfers* progressing through a communication channel. The most simple message transmission in the shared memory channel ($c = 0$)[1] is done through an intermediate buffer placed in a shared memory area. This is the common mechanism adopted by the widespread MPI implementations MPICH and Open MPI for transferring small messages. If m is the size of the message, the transmission starts after a time represented by the *overhead* parameter $o^c(m)$, defined as the time elapsed since the invocation of a message transmission operation until the beginning of data injection into the channel. The transmission requires two transfers, with a total cost

$$T^0_{p2p}(m) = o^0(m) + 2L^0(m,1) \quad (1)$$

The first transfer copies the data to the intermediate buffer, with cost $L^0(m,1)$. When data is ready in the intermediate buffer, the receiver starts the second transfer to its user buffer. The parameter *transfer time* L is the cost of a transfer and is represented as $L^c(m,\tau)$. As the transfer may flow concurrently with others, its cost depends not only on the message size m, but also on the number τ of such *concurrent transfers*.

In the network channel ($c = 1$), τ–Lop follows the same scheme by Cameron et al. in [31], that considers a simple message transmission as composed of three transfers: first, from the sender buffer to the internal buffer of the NIC in the sender node, second, from here to the NIC in the receiver node, and last, from here to the receiver buffer. τ–Lop considers the first and third transfers as shared memory transfers, with cost L^0, whereas the second transfer progresses through the network, with cost L^1. Thus, the cost of a point-to-point transmission through channel c^1 becomes

$$T^1_{p2p}(m) = o^1(m) + 2\,L^0(m,1) + L^1(m,1) \quad (2)$$

Nevertheless, the number of transfers composing a network transmission depends on the network technology. For instance, some networks, e.g. Infiniband, allow the direct transfer to the user buffer of the receiver using a Remote Direct Memory Access mechanism [32], hence:

$$T^1_{p2p}(m) = o^1(m) + L^1(m,1) \quad (3)$$

The main distinguishing feature of τ–Lop is its ability to capture the fact that the contention for the channel increases the cost of each individual transfer. This effect in both shared memory and networks is deeply studied in [29], [30] and [33]. Briefly exposed, τ–Lop also represents the cost of A concurrent transfers with the $\|$ operator, so that $A \,\|\, L^c(m,1) = L^c(m,A)$ and more generally $A \,\|\, L^c(m,\tau) = L^c(m, A \times \tau)$. The operator is then extended so that $A \,\|\, T^c(m)$ represents the cost of A concurrent transmissions of a message of size m contending for the channel c. To be evaluable, expressions involving transmissions must be reduced to expressions of just transfer times and overheads. For instance $2 \,\|\, T^0_{p2p}(m) = 2 \,\|\, \left[o^0(m) + 2L^0(m,1)\right] = o^0(m) + 2L^0(m,2)$. Note that the overhead cost is attributable to the processor, a non-shared resource, and hence not affected by $\|$.

1. The communication channels in a system are identified by a number c starting from 0. Zero is the number usually assigned to the channel with the best overall performance.

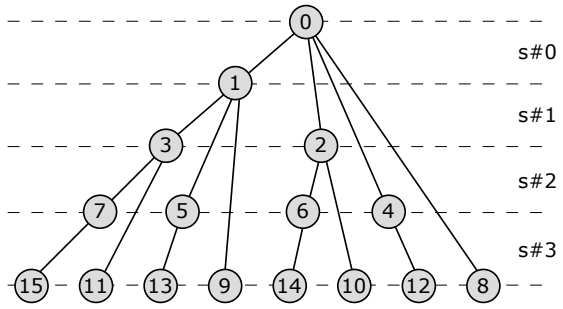


Fig. 4: An Open MPI Binomial Tree with $P = 16$ processes.

The definition of L enforces the restriction that the cost of A concurrent transfers will be between the cost of a single transfer and that of A consecutive ones, that is, $L(m,1) \leq L(m,A) \leq A \times L(m,1)$, an expression that can be generalized as $L(m,\tau) \leq L(m, A\times\tau) \leq A \times L(m,\tau)$. In addition, the transfer time cost grows linearly with the increase of the message size, and hence $L(A \times m, \tau) = A \times L(m,\tau)$. Once the τ–Lop parameters have been calculated, expressions (1)-(3) are applied to estimate the cost of transmissions.

The concept of *concurrent transmissions*, those sharing the channel, allows τ–Lop to better represent collective operations. In fact, τ–Lop was initially conceived to model more accurately the cost of collectives on homogeneous systems. In general, a collective operation executes in a sequence of stages, where the number of involved processes and the message size may change along the stages. For instance, the *Binomial Tree* algorithm used in the *MPI_Bcast* collective, represented in Fig. 4, runs in $\lceil \log_2 P \rceil$ stages, the height of the tree. The cost of the operation is given by:

$$\Theta_{Bin}(m) = \sum_{i=0}^{\lceil \log_2(P) \rceil - 1} \left[2^i \,\|\, T^c(m)\right] \quad (4)$$

In the first stage, the rank *root* sends the message of size m to the rank $root + 2^i$. The algorithm recursively continues with both processes acting as roots of sub-trees with $P/2$ processes. The number of sending processes doubles in each stage, whereas the message size remains constant. The key point, however, is that τ–Lop clearly states how contention grows exponentially in each stage. Formula (4) gives a good insight into the τ–Lop expressive power, particularly at the transmission level.

Regular τ–Lop analyses in homogeneous platforms lead to tractable cost expressions in the forms $n \,\|\, T^c(m)$ and $T^c(m_1) + T^c(m_2)$. Heterogeneous environments, however, give place to more complex, even intricate, cost formulas, as those in the form $T^{c_1}(m_1) \,\|\, T^{c_2}(m_2)$ for different c_1 and c_2 channels. The formal development of such expressions causes an impasse in the decomposition path, whose breaking imposes an extension of τ-Lop. Such extension consists of the following three assumptions, illustrated in Fig. 5.

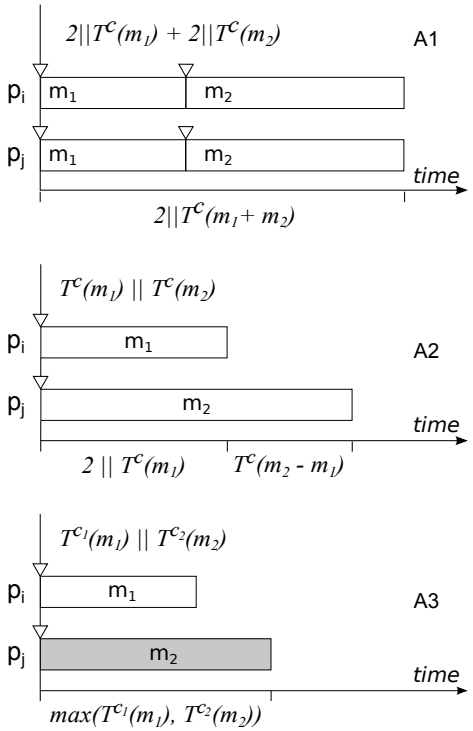


Fig. 5: Graphic representation of the three assumptions defining the τ–Lop extension for heterogeneous platforms.

A1. A sequence of transmissions progressing through the same channel has the cost of a single transmission of a message of aggregate size:

$$A \,\|\, T^c(m_1) + A \,\|\, T^c(m_2) = A \,\|\, T^c(m_1 + m_2)$$

The expression assumes the error of disregarding the overhead of second and subsequent transmissions, so its application can be canceled for small messages, where overhead is a significant term.

A2. Two message transmissions through the same communication channel progress concurrently during the transmission time of the shorter one:

$$T^c(m_1) \,\|\, T^c(m_2) = 2 \,\|\, T^c(m_1) + T^c(m_2 - m_1), m_2 \geq m_1$$

A3. Two transmissions progressing through different communication channels do not interfere. The total cost is the maximum of the individual costs:

$$T^{c_1}(m_1) \,\|\, T^{c_2}(m_2) = max\{T^{c_1}(m_1), T^{c_2}(m_2)\}$$

Once the extension has been stated, we show next an example of use. Expression (5) appears quite often for heterogeneous kernels, as we will see in the next section.

$$[T^{c_0}(m_a) + T^{c_1}(m_b)] \parallel [T^{c_0}(m_c) + T^{c_1}(m_d)] \qquad (5)$$

It models the concurrence of two sequences of transmissions through different communication channels, and leads to different developments depending on the size of the messages. The most simple case happens when $m_a = m_c$ and $m_b = m_d$. Then, the transmissions through the channel c_0 progress concurrently, as the c_1 transmissions do, and hence we have two concurrent pairs of transmissions on which we can apply assumption $A2$ as:

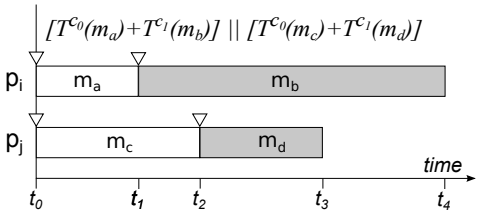


Fig. 6: Graphic representation of τ–Lop modeling of a complex expression commonly found in heterogeneous platforms. Each process transmits a sequence of two messages through two different communication channels c_0 and c_1. Transmissions have different costs.

$$[T^{c_0}(m_a) \,\|\, T^{c_0}(m_c)] + [T^{c_1}(m_b) \,\|\, T^{c_1}(m_d)] = 2 \,\|\, T^{c_0}(m_a) + 2 \,\|\, T^{c_1}(m_b)$$

Fig. 6 illustrates a more general case of (5), where $m_c \geq m_a$, $m_b \geq m_d$, and $T^{c_1}(m_b) \geq T^{c_0}(m_c - m_a)$. At time t_0 we have two concurrent transmissions of messages m_a and m_c through the same channel c_0. By $A2$, (5) becomes

$$2 \,\|\, T^{c_0}(m_a) + (T^{c_1}(m_b) \,\|\, [T^{c_0}(m_c - m_a) + T^{c_1}(m_d)]) \,. \quad (6)$$

The first term $2 \,\|\, T^{c_0}(m_a)$ represents the cost between times t_0 and t_1. The second term begins with the cost of two transmissions starting at time t_1 progressing through different channels, with individual costs $T^{c_1}(m_b)$ and $T^{c_0}(m_c - m_a)$. Applying $A3$ we have a joint cost $max\{T^{c_1}(m_b), T^{c_0}(m_c - m_a)\}$, which is $T^{c_1}(m_b)$ by hypothesis. In other words, $T^{c_0}(m_c - m_a)$ drops out from (6), resulting in $2 \,\|\, T^{c_0}(m_a) + [T^{c_1}(m_b) \,\|\, T^{c_1}(m_d)]$, where we have two transmissions that progress concurrently through c_1. By applying $A2$ to the second term, (5) finally becomes

$$2 \,\|\, T^{c_0}(m_a) + 2 \,\|\, T^{c_1}(m_d) + T^{c_1}(m_b - m_d). \qquad (7)$$

4 Modeling Real Common Kernels

To test the abilities of the τ–Lop extensions, we have hybridized two data-parallel kernels of rather opposite features, namely the SUMMA matrix multiplication algorithm, with a high communication to computation ratio, and a finite difference solver of the 2D wave equation, much less demanding in terms of communication load.

4.1 The SUMMA Algorithm

The Scalable Universal Matrix Multiplication Algorithm (SUMMA) [34] is a data-parallel kernel that is present in many scientific applications. It can be found, for example, in the linear algebra ScaLAPACK library. This subsection discusses its τ–Lop cost as a hybrid case study.

SUMMA involves a set of processes that cooperatively compute the dense matrix multiplication $C = A \times B$. For simplicity, square matrices are supposed. The elements of the matrices are grouped into blocks of size $b \times b$. The size of the matrices is then $\mathcal{N} = N \times N$ blocks. The block is the unit of computation. Let us first consider a homogeneous system. Fig. 7 shows an example for $P = 16$ processes. The blocks are distributed evenly between the processes following a 2D arrangement, hence balancing both the computational load and the communication volume of each process. SUMMA

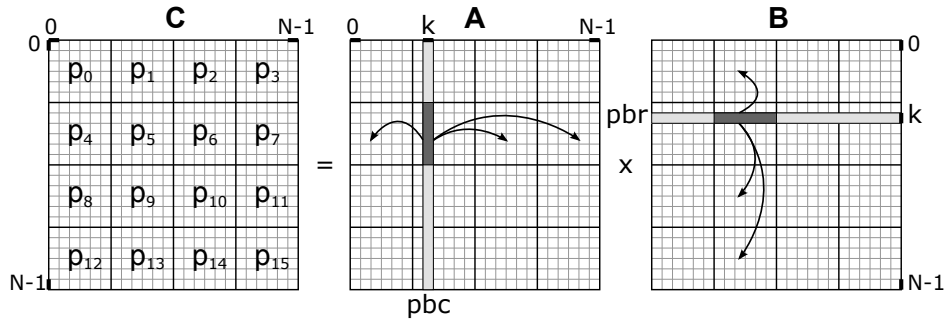


Fig. 7: Partition of the matrix multiplication SUMMA algorithm in a homogeneous platform between $P = 16$ processes. A rectangle of 6×6 blocks is assigned to each process. The kth iteration is shown. Processes with blocks in the pivot block column (*pbc*) of matrix A (p_1, p_5, p_9 and p_{13}) and in the pivot block row (*pbr*) of B (p_4, p_5, p_6 and p_7) transmit their blocks to the rest. For instance, in the iteration showed k, process p_5 sends its part of the *pbc* to the processes in the same row (p_4, p_6 and p_7), and then it sends its part of the *pbr* to the processes in the same column (p_1, p_9 and p_{13}).

executes in N iterations. A column and a row of blocks traverse the matrices A and B respectively with each iteration, from $k = 0$ to $N - 1$. They are called the pivot block column (*pbc*) and pivot block row (*pbr*). In each iteration k, the processes compute partial results for all of their assigned blocks of C so that $c_{i\,j}$ becomes $c_{i\,j} + a_{i\,k} \times b_{k\,j}$. To this end, each process has to receive the k-th column block a_{ik} of A and the k-th row block b_{kj} of B. After N iterations, each block of matrix C will have the value $c_{i\,j} = \sum_{k=0}^{N-1} a_{i\,k} \times b_{k\,j}$. Thus, each iteration k is composed of three stages[2], namely (I) the processes owning the k-th *pbc* of the matrix A send the blocks to the processes in the same row, (II) the processes owning the k-th *pbr* of the matrix B send the blocks to the processes in the same column, and (III) each process p_i updates the blocks in its assigned rectangle of matrix C.

In heterogeneous platforms, the number of blocks assigned to each process depends on its speed. Hence, the size of the rectangles is not homogeneous, and the communication pattern changes. Fig. 8 shows an example. Following the 2D partitioning algorithm by Beaumont et al. in [36] (see Fig. 3), the processes are arranged in columns. Next the cost $\Theta^{(80)}$ of the iteration $k = 80$ is modeled using τ-Lop. Two computing nodes are considered, and hence, two communication channels, shared memory ($c = 0$) and network ($c = 1$). The cost is modeled under the assumption that all processes start the communication phase of the iteration at the same time. Note that in the iteration 80 the blocks of the *pbc* are owned and hence sent by processes p_1 and p_4. Looking carefully at Fig. 8 we can appreciate that p_1, for instance, sends 134 blocks to p_0. According to it and to assumption $A1$, the cost of the transmissions of p_1 is $T^0(134) + T^1(158)$. Similarly, that of p_4 is $T^0(116) + T^1(104)$. The total cost, taking into account the simultaneity of the communications in the rows, is $\big(T^0(134) + T^1(158)\big) \parallel \big(T^0(116) + T^1(104)\big)$, which fits the pattern of expression (5), hence becoming $\big(T^0(134) \parallel T^0(116)\big) + \big(T^1(158) \parallel T^1(104)\big)$. Finally, by applying assumption $A2$, we arrive to the following cost:

$$\Theta^{(80)}_{pbc} = 2 \parallel T^0(116) + T^0(18) + 2 \parallel T^1(104) + T^1(54) \quad (8)$$

Transmissions of the *pbr* in each column of the matrix B progress concurrently. Furthermore, note that they progress through the network channel. The first column has a communication cost of $T^1(124)$ (from p_1 to p_4), the second column has a cost of $T^1(97)$ (from p_0 to p_5), and the third column cost is $T^1(35)$ (from p_3 to p_2). The total cost of sending the *pbr* will be hence $T^1(124) \parallel T^1(97) \parallel T^1(35)$. The application of $A2$ yields:

$$\Theta^{(80)}_{pbr} = 3 \parallel T^1(35) + 2 \parallel T^1(62) + T^1(27) \quad (9)$$

The total cost in the iteration $k = 80$ is the addition of (8) and (9), that is, $2 \parallel T^0(116) + T^0(18) + 2 \parallel T^1(104) + T^1(54) + 3 \parallel T^1(35) + 2 \parallel T^1(62) + T^1(27)$. Bringing terms together we have $3 \parallel T^1(35) + (\,2 \parallel T^1(104) + 2 \parallel T^1(62)\,) + (\,T^1(54) + T^1(27)\,) + 2 \parallel T^0(116) + T^0(18)$. Finally, applying A1:

$$\begin{aligned} \Theta^{(80)} = 3 \parallel T^1(35) + 2 \parallel T^1(166) + T^1(81) \\ +2 \parallel T^0(116) + T^0(18) \end{aligned} \quad (10)$$

The algorithm total cost is

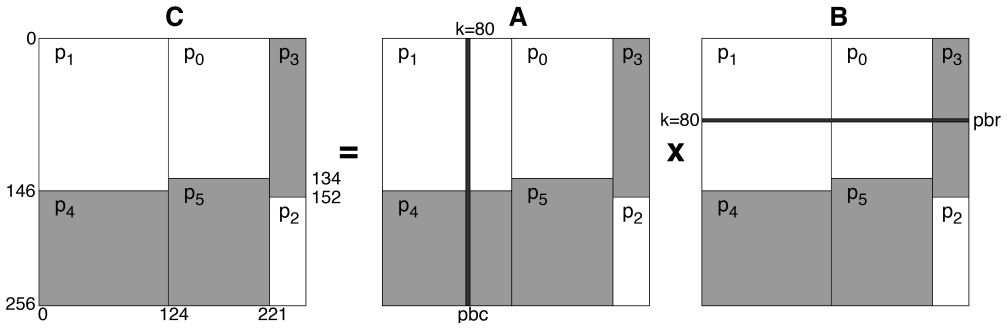
$$\Theta_{SUMMA} = \sum_{k=0}^{N} \Theta^{(k)}. \quad (11)$$

Note that any change in the arrangement of the rectangles of Fig. 8 leads to a different communication cost. Malik [37] proposes some heuristics to reduce inter-node communications by relocating the intra-column rectangles.

The estimation of the HLogGP cost of SUMMA is formulated as follows. For vertical communication, we calculate the total cost of every column, and we take the maximum, because the communication progresses in parallel. For the horizontal communication, we take the maximum cost of the processes in each column block. The total cost is the addition of both horizontal and vertical costs [3].

2. The communication performance can be improved using collective and non-blocking communication operations, allowing the overlapping of communication and computation. Other enhancements have been proposed, as the HSUMMA algorithm by Hasanov et al. [35], which proposes a hierarchical approach for the communications.

3. The implementation of the micro-benchmarks and the tool for estimating the communication costs are available on line ([38]), and their description as a supplementary material.



pbc communication in iteration 80

Sender	Receiver	τ–Lop cost
p_1	p_0	$T^0(134)$
p_1	p_3	$T^1(146)$
p_1	p_5	$T^1(12)$
p_4	p_2	$T^1(104)$
p_4	p_3	$T^0(6)$
p_4	p_5	$T^0(110)$

Fig. 8: The SUMMA algorithm on a heterogeneous platform, with $P = 6$ and $M = 2$. A rectangle of different size is assigned to each process. White rectangles are assigned to processes running on the node 0, and grey rectangles to that on the node 1. The figure shows the iteration $k = 80$. Now p_1 sends its part of the *pbc* to p_0, p_3 and p_5. We say that p_1 *overlaps* them. Likely p_1 sends its part of the *pbr* to the processes in the same column (p_4). The *pbc* transmissions involved and their associated cost estimations are shown at the right side, where the message size is expressed in terms of blocks.

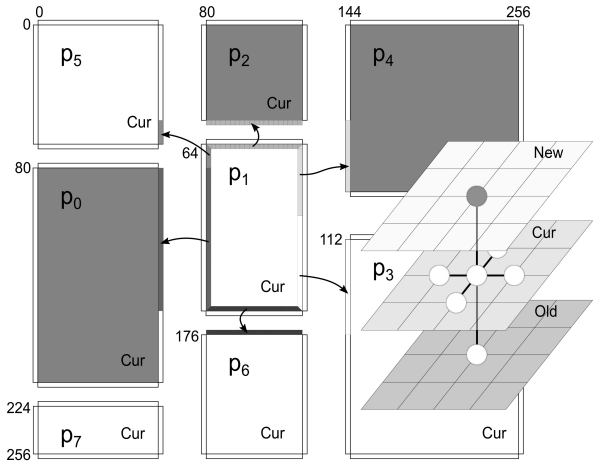


Fig. 9: A 2D partition for solving the wave equation in a mesh of $N = 256$ doubles. Each process recomputes its rectangles New, Cur and Old along time. The computation stencil imposes communications in Cur. Process p_1 sends its perimeter elements of Cur, which will form the halo of its neighbors.

4.2 The 2D Wave Equation

The technique of finite differences is also ubiquitous in HPC. We use it to build a hybrid kernel that solves the 2D wave equation, formulated as $\frac{\partial^2 u}{\partial t^2} = c^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$ with initial conditions $u(x, y, 0) = I(x, y)$ and $\frac{\partial}{\partial t} u(x, y, 0) = 0$. The discrete solution $u(x, y, t)$ is approached in the mesh $x \in (0, N)$, $y \in (0, N)$ and $t \in (0, T]$. In our implementation, we use the boundary conditions $u(0, y, t) = u(x, 0, t) = u(N, y, t) = u(x, N, t) = 0$. Along time, $u(x, y, t+1)$ is given by successive instances of matrix New, generated from matrices Cur ($u(x, y, t)$) and Old ($u(x, y, t-1)$) according to the recursive finite differences algorithm driven by the stencil at the right of Fig. 9:

$$\begin{aligned} New(i,j) = {} & 2(1-2C^2)Cur(i,j) - Old(i,j) \\ & + C^2 Cur(i-1,j) + C^2 Cur(i+1,j) \\ & + C^2 Cur(i,j-1) + C^2 Cur(i,j+1) \end{aligned} \quad (12)$$

Note that (12) imposes the communication of perimeter elements between the Cur rectangles. Fig. 9 illustrates a 2D partition between eight processes running on two machines. The transmissions from p_1 to its neighbors are shown. For instance, if η_i denotes the neighborhood of p_i in the clockwise order, then $\eta_1 = \{2, 4, 3, 6, 0, 5\}$. An alternative 1D partition of the data space in matrix slices would lead to a more simple communication scheme, where each process has just two neighbor processes for interchanging data. In this paper, we evaluate the cost derived from 2D partitions. Non-blocking sends (and receives) are used, so all transmissions start at once. The cost per iteration allocated to p_i is

$$\Theta_i = \mathop{\Vert}_{j \in \eta_i} T^{c(j)}(m(j)), \quad (13)$$

where $m(j)$ is the size of the message sent to neighbor p_j, and $c(j)$ is the channel used to send the message. For p_1, in terms of doubles, $\Theta_1 = T^1(64) \,\Vert\, T^1(48) \,\Vert\, T^0(64) \,\Vert\, T^0(64) \,\Vert\, T^1(96) \,\Vert\, T^0(16)$. As the transmissions of all P processes progress concurrently, the cost of each iteration is

$$\Theta = \left[\mathop{\Vert}_{i=0}^{P-1} \Theta_i \right] \quad (14)$$

The τ–Lop cost of the algorithm is hence $\Theta_{w2D} = T \times \Theta$.

HLogGP, however, estimates the cost per process as the sum of the transmission costs to the neighbors. The iteration cost is the maximum of the per process cost.

5 Model Evaluation

In this section we measure the real-life communication cost of a broad set of sixteen hybrid configurations of SUMMA and Wave2D kernels in Fermi. These figures are compared with their τ–Lop and HLogGP estimations. For the purpose of reproducibility, before presenting the results, we detail the heterogeneous test platform, the procedure to estimate the τ–Lop and HLogGP parameters, discuss some issues on the generation of the test *configurations* and show the way of measuring the real communication times.

5.1 The Heterogeneous Test Platform

The platform, called *Fermi*, is equipped with 16 computing nodes. Nine nodes have two six-core Intel Xeon E5649 processors (2.53 GHz), while the other seven have two quad-core Intel Xeon E5520 processors (2.27 GHz), making 108+56

= 164 cores. Two NVidia Tesla M2075 GPUs are attached to each of the 12-core nodes, while two C2050 GPUs to each of the 8-core nodes. The nodes have a QDR Infiniband (40 Gbps) and a TCP/Ethernet (1 Gbps) network interfaces.

Operating system is CentOS 6.5, with libraries for computation and communication. In the CPUs, a process uses the function *dgemm* of the Intel MKL library to compute the SUMMA kernel on a rectangle of double precision elements, and the implementation of the recursive finite differences algorithm in (12) for the Wave2D kernel. OpenMP threads are used for multi-core processes. In the GPUs, cuBlas library for SUMMA and a self made kernel for Wave2D are used with the same purpose.

Open MPI 1.8.1 is used for communication. In SUMMA, row communication uses the non-blocking primitives, *MPI_Isend* and *MPI_Irecv*, while column communication uses blocking primitives *MPI_Send* and *MPI_Recv*. This is a standard SUMMA implementation, though other options are possible including blocking communication for *pbc*, broadcast, pipeline or ring *pbr*, etc. Non-blocking communication primitives are used in Wave2D.

5.2 Parameter Measurement of τ–Lop and HLogGP

The parameter set of both τ–Lop and HLogGP is next calculated for shared memory, TCP and Infiniband (IB) channels. The procedure to measure the τ–Lop parameters is detailed in [30]. Let M_t be the number of different types of nodes in the system, with $M_t = 2$ in Fermi.

A $Ring_\tau^0$ operation is defined for calculating the *Transfer Time* parameter $L^0(m,\tau)$ of the shared memory channel. In the operation, process p_i sends a message to process p_{i+1} and receives from process p_{i-1}. $Ring_\tau^0$ is run in each type of node, for a range of message sizes m and increasing number of processes concurrently communicating τ. Similarly, a $Ring_\tau^1$ operation is set for calculating the network parameters $L^1(m,\tau)$, with the processes distributed in *Round Robin* in two nodes of the system. With M_t types of nodes, the operation is executed $M_t^2 - \binom{M_t}{2}$ times for all the possible combinations. The number of experiments performed is hence $M_t + R_t(M_t^2 - \binom{M_t}{2})$, been R_t the number of network types used. In Fermi, for instance, using shared memory and Infiniband ($R_t = 1$) is $2 + 1 \times (2^2 - \binom{2}{2}) = 5$.

An important feature of τ–Lop is that the number of experiments needed to estimate the parameter values is of order M_t^2. LMO and HLogGP shows order M^2, usually much higher. In addition, in both HLogGP and τ–Lop, the parameters are estimated with independence of the final distribution of the processes in the platform, and therefore, they do not need to be estimated for each configuration.

For HLogGP, we implemented the parameter measurement micro-benchmarks detailed in [23] and found them rather simplistic resulting in low accuracy in the estimations of the overhead and gap per message parameters, showing a high variance. In any case, average values are taken.

5.3 Building the Configurations

The sixteen configurations used in the evaluation depart from the set of sixteen process layouts of Table 1, listed by growing number of processes P. The M column splits the number of nodes in a layout by type, 12-core nodes and 8-core nodes. M13, for instance, uses nine 12-core nodes and four 8-core nodes. CPU processes execute in one or more cores using OpenMP threads. Columns $c = \kappa$ indicate the number of processes in the row with κ cores assigned. The point here is that having processes with different computational capabilities increases the heterogeneity of the configuration. Finally, GPU indicates the number of processes running on GPU. They use a dedicated core in the node for MPI communication and management of the data transfers between host and GPU memories. The cost of these transfers is included in the FPM of a GPU process [39]. Note that the P column is the summation of the former six.

Fig. 2 showed how a configuration comes from a process layout and a balanced data partition. The procedure to obtain this last one, using FuPerMod, is described in section 2. An user-provided benchmark feeds FuPerMod. The computation of expression (12), for instance, is the benchmark used in the Wave2D kernel. This benchmark is executed

TABLE 1: Characterization of a set of sixteen nodes layouts used in the evaluation. They are provided by the SLURM scheduler of Fermi.

Name	M	c=6	c=5	c=4	c=3	c=2	GPU	P
M1	1+0	2	0	0	0	0	0	2
M2	2+0	3	1	0	0	0	1	5
M3	2+1	3	1	2	0	0	1	7
M4	3+1	4	1	3	0	0	3	11
M5	3+2	4	1	4	1	0	4	14
M6	3+3	4	1	5	1	1	6	18
M7	4+3	6	1	5	1	1	6	20
M8	8+0	11	3	2	0	0	7	23
M9	9+0	12	3	3	0	0	9	27
M10	9+1	12	3	5	0	0	9	29
M11	9+2	12	3	6	1	0	10	32
M12	9+3	12	3	7	1	1	12	36
M13	9+4	12	3	9	1	1	12	38
M14	9+5	12	3	10	2	1	13	41
M15	9+6	12	3	11	2	2	15	45
M16	9+7	12	3	13	2	2	15	47

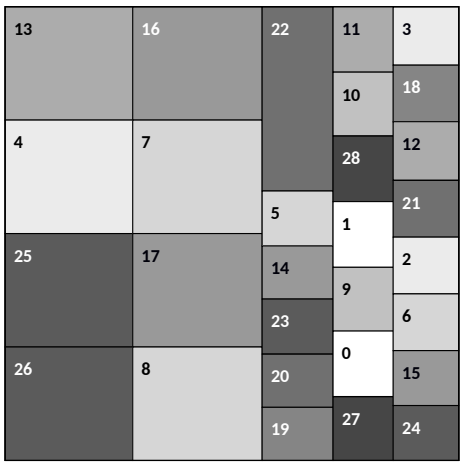


Fig. 10: SUMMA data partition of the $M10$ configuration. The number indicates the MPI rank for a total of $P = 29$ processes. The fill pattern indicates the node where the process executes, for a total of $M = 10$ nodes. Note how the partition algorithm arranges the rectangles in columns.

by all the processes of a given layout. The produced FPMs are then used to generate the balanced partition of the data matrix into block rectangles, one per process.

It is worth noting that the size of a rectangle must be kept within certain limits in the interest of a correct design of the evaluation experiments. If a rectangle has $x \times y$ blocks, we have to compute them, communicate x pivot row blocks and y pivot column blocks. In short, the computation time grows quadratically while the communication time grows linearly. As we need to make a meaningful addition of the two times, the ratio of computation to communication times has to be acceptable, which imposes test matrix sizes, and hence N/P values, under certain bounds. The chosen sizes are $N = 128$, $N = 256$ and $b = 32$ doubles, which yield matrices of $(4K)^2$ and $(8K)^2$ elements respectively, distributed between 2 and 47 processes. Fig. 10, for instance, shows the balanced data partition obtained from layout $M10$.

Algorithm 1 Measuring the cost of the SUMMA kernel

```
MPI_Comm_rank(WORLD, ↑me)
for n = 0 to N − 1 do
  MPI_Barrier(WORLD)
  t_start ← MPI_Wtime()

  // Horizontal communication
  for all p ∈ 0..P − 1 do
    if overlap(me, p) then
      if hold_pbc(me, n) then
        MPI_Type_vector(. . .)
        MPI_Pack(packed_pbc ← pbc)
        MPI_Isend(packed_pbc, overlap_sz(n, me, p),
              MPI_DOUBLE, p, TAG, WORLD, ↑req[p])
      else
        MPI_Irecv(pbc, overlap_sz(n, me, p),
              MPI_DOUBLE, p, TAG, WORLD, ↑req[p])
      end if
    end if
  end for

  // Vertical communication
  if hold_pbr(me, n) then
    MPI_Comm_size(vComm, ↑Q)
    for all q ∈ 0..Q − 1, q <> self do
      MPI_Send(pbr, col_width, MPI_DOUBLE,
            q, BTAG, vComm)
    end for
  else
    MPI_Recv(pbr, col_width, MPI_DOUBLE,
          pbr_holder, BTAG, vComm, ST_IGNORE)
  end if
  MPI_Waitall (WORLD, req, ST_IGNORE)
  t_end ← MPI_Wtime()

  // Computation
  dgemm(. . . )
end for
```

5.4 Empirical Measurement of Communication Times.

Once a target configurations have been built, every and each of its process executes the full kernel code, which access the configuration as needed. Algorithm 1 is the skeleton of SUMMA kernel. Following, we discuss some aspects of this code, though similar considerations can be directly applied Wave2D. Non-blocking communication is used for the horizontal transmission of the *pbc*. Each process determines if any other process *overlaps* it (see Fig. 8). If this is the case, it will send or receive the portion of the *pbc*. Note that the *pbc* blocks are not contiguous in memory, so the sender needs to pack them before sending. Our extensive experiments have shown that packing time is negligible with respect to transmission time. Regarding the *pbr*, it is sent using blocking primitives. As all rectangles in the same column have the same width in terms of blocks, the communicating processes are known in advance (those in the column). This fact allows us to avoid the search for overlaps and create one communicator per column (*vComm*) at initialization time.

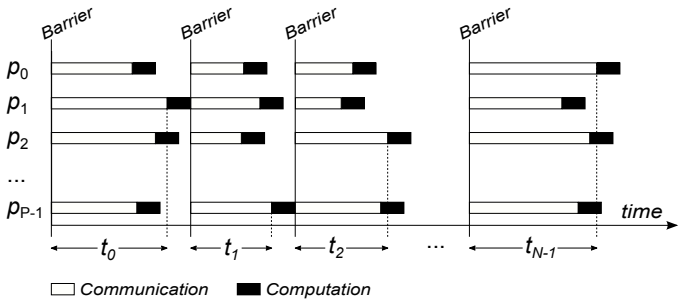
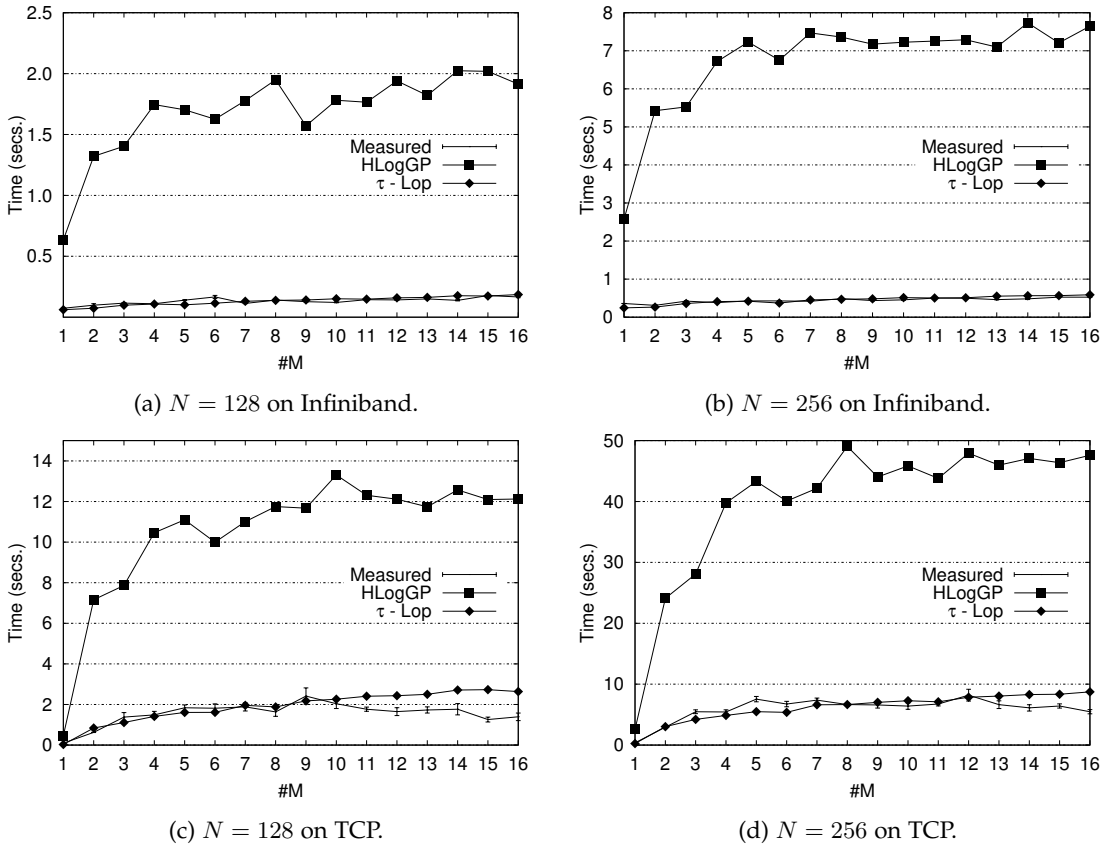


Fig. 11: Methodology proposed to measure the empirical communication cost of a data-parallel kernel. The length of a white bar represents the cost of process p in the iteration n. The global cost of the iteration n is t_n, defined as the maximum cost of the P processes.

SUMMA, and any other data-parallel kernel, has a common execution pattern that repeats cycles of communication and computation. Each iteration performs a partial block computation after completing the communications. The methodology adopted in SUMMA to measure the cost of the communication stage (again extensible to any other 2D kernel) assumes that all processes arrive at the same time to the communication stage, a limitation imposed by the models to give more accurate predictions. This assumption isolates communication from computation to a great degree, ensuring that the communication time is measured quite accurately. The FuPerMod best efforts to produce a balanced partition greatly helps to accomplish the goal. Still, as some computation imbalances may exist between the processes, we call $MPI\_Barrier$ at the beginning of each iteration. Fig. 11 helps to understand this framework. The measured time of iteration n is t_n, defined as the maximum of the $t\_end - t\_start$ figures of the P processes. The total real-life SUMMA cost is $t = \sum_{n=0}^{N-1} t_n$, next compared to the analytical cost expression (11).

5.5 Comparison of Estimated and Measured Costs.

We have developed a tool that automatically calculates the τ–Lop cost estimation of a kernel for a given configuration [38]. The tool inputs are the configuration and the set of τ–Lop parameters of the platform, as shown at the bottom right side Fig. 2. A similar tool for HLogGP is also provided.



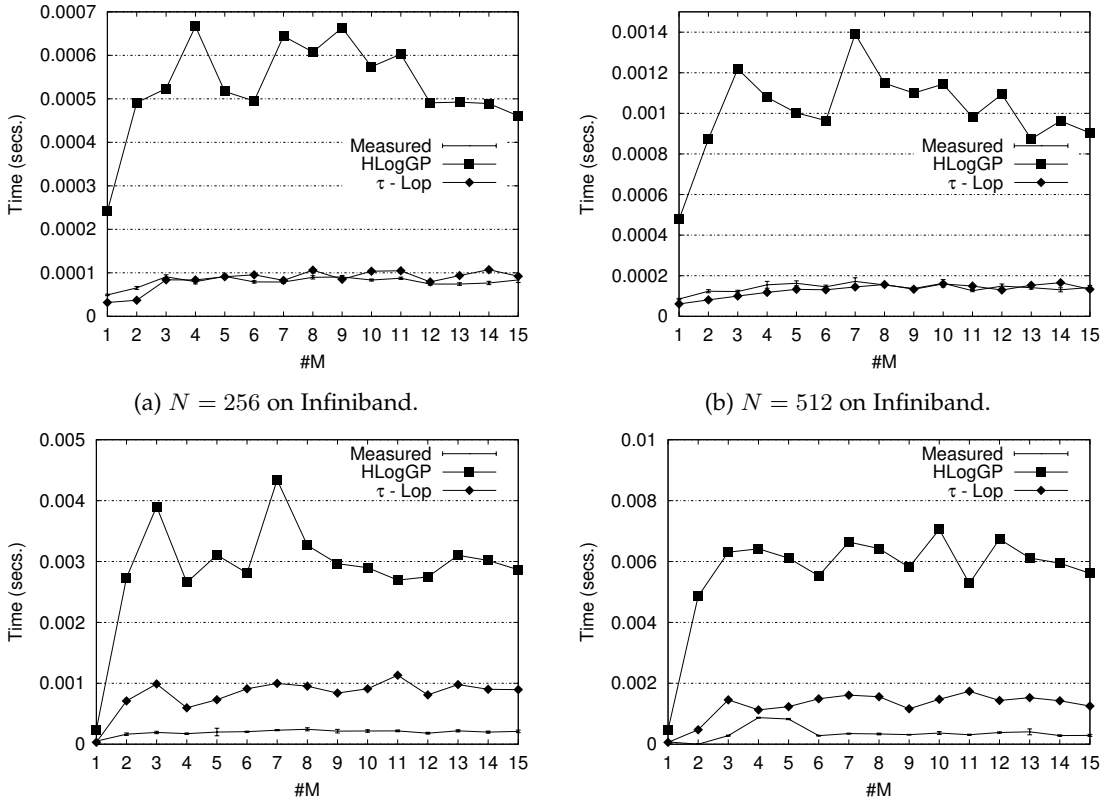
(a) $N = 128$ on Infiniband.

(b) $N = 256$ on Infiniband.

(c) $N = 128$ on TCP.

(d) $N = 256$ on TCP.

Fig. 12: SUMMA measurements versus estimations for different configurations, networks and matrix sizes.



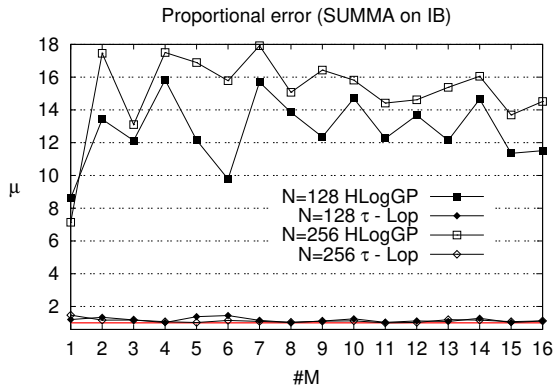
(a) $N = 256$ on Infiniband.

(b) $N = 512$ on Infiniband.

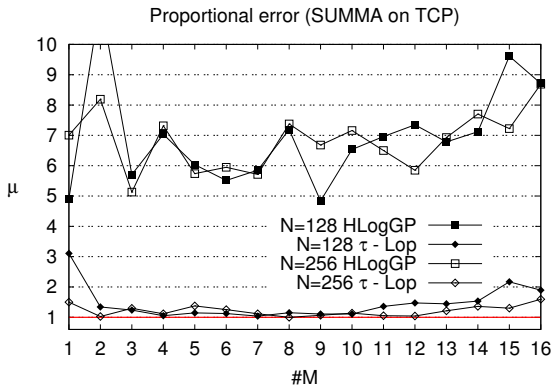
(c) $N = 256$ on TCP.

(d) $N = 512$ on TCP.

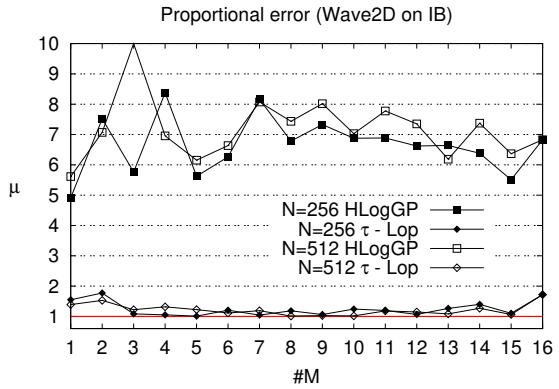
Fig. 13: Wave2D measurements versus estimations for different configurations, networks and matrix sizes.



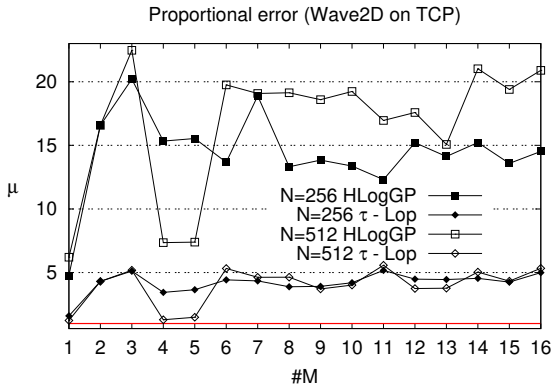
(a) SUMMA on Infiniband Proportional error.



(b) SUMMA on TCP Proportional error.



(c) Wave2D on Infiniband Proportional error.



(d) Wave2D on TCP Proportional error.

Fig. 14: Data of Fig. 12 and Fig. 13 expressed in terms of Proportional error.

Fig. 12 plots the comparison between estimated and measured costs of the SUMMA configurations, of increasing complexity from left to right. Though hard to appreciate, the measured costs include the typical deviation for multiple executions of the kernel. The left side plots are built with $N = 128$, while $N = 256$ applies to the right side plots. The two upper plots use the IB network and the two lower plots use TCP. Note the scale difference between them (up to one order of magnitude) due to the Infiniband higher performance. Likewise, Fig. 13 applies to the Wave2D kernel, with $N = 256$ and $N = 512$.

Interesting enough, one may observe that the measured communication time remains almost constant along the configurations, that is, with growing number of processes. The reason is that each process has to transmit less data to the other processes (a smaller rectangle implies a smaller number of elements per process), but it has to perform a higher number of communications. τ–Lop accurately captures this fact on these highly complex and diverse scenarios. The error plots reveal two points that need to be emphasized. First, the error does not change significantly between different matrix sizes and networks, which shows the *robustness* of the model, with one exception, TCP estimations for the Wave2D kernel, discussed later. Second, the error remains almost constant for the range of configurations, what highlights that the model keeps *scalable* in hybrid platforms.

HLogGP predictions are far from the measured costs in both kernels, and show a high variability between configurations. In our opinion, the main reasons are that the model (1) proposes a methodology of parameter measurement based on micro-benchmarks that is not accurate, (2) has not been validated for communication patterns beyond the simple master-worker transmissions, and (3) does not consider the contention in communication.

We use the *Proportional error* to express the accuracy of estimation with respect to the real measurement, instead of the *Relative error* $\rho = |e - r|/r$ usually found in the literature, where e and real r are the estimated and real values respectively. The proportional error ρ suffers from an anomaly: an underestimation gives a lower relative error than the overestimation in the same proportion. This effect skews significantly the interpretation of our results. Therefore, we set aside the relative error ρ in favor of the proportional error $\mu = max(r, e)/min(r, e)$, which is always greater than 1 on error, and equal to 1 otherwise. Notwithstanding, a simple relation between ρ and μ exists that allows the conversion from one another [30]. Fig. 14 shows the *Proportional error* of Fig. 12 and Fig. 13. For both SUMMA and Wave2D, τ–Lop shows a good accuracy for all experimental settings, except for the case of TCP Wave2D. We attribute this behavior to much shorter messages used in the Wave2D application. While τ–Lop predictions are consistently accurate for longer messages (bigger matrices) for both Infiniband and TCP, we observe some drop in accuracy for short messages in TCP networks. We have to improve the measurement method of the τ–Lop parameters and take

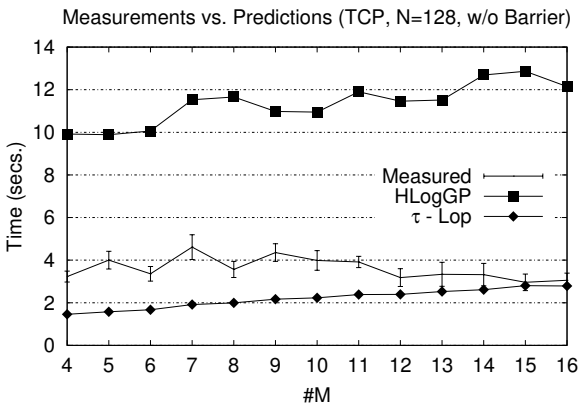


Fig. 15: Communication measurements versus predictions for a range of configurations in TCP network and $N = 128$, without using `MPI_Barrier` in Algorithm 1.

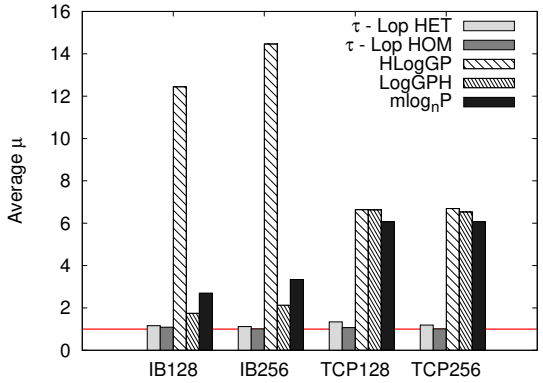


Fig. 16: Average Proportional Error on the SUMMA kernel made by heterogeneous τ-Lop with respect to homogeneous τ-Lop, by HLogGP with respect to LogGPH, and by mlog$_n$P.

into account the TCP network behavior for short messages, including the piggy-backing mechanism. At the same time,

TABLE 2: Average Proportional error $\bar{\mu}$ along the configurations of Fig. 14.

		SUMMA		Wave2D	
Network	Model	N=128	N=256	N=256	N=512
Infiniband	τ–Lop	1.16	1.12	1.21	1.20
	HLogGP	12.44	14.46	6.53	7.06
TCP	τ–Lop	1.34	1.19	3.88	3.11
	HLogGP	6.64	6.69	13.08	14.19

HLogGP poorly predicts the communication time in most cases, also showing significant and irregular fluctuations in the accuracy of prediction. The Table 2 summarizes these experimental results.

The models assume that processes start the communication phase at the same time. Therefore, to fairly estimate the accuracy of their predictions, we used barriers in the beginning of the communication phase in our applications. In order to see how the predictive accuracy changes if we remove the barriers, we also experimented with the SUMMA kernel without the `MPI_Barrier` synchronization. As Fig. 15 shows, the lack of synchronization of the processes in each phase worsens the τ–Lop predictions. Nevertheless, the applications can still benefit from much more accurate τ–Lop predictions of the communication time. Interestingly, the communication time is longer than that of the kernel using the barrier synchronization (see Fig. 12c), a behavior reproduced in all the configurations, matrix sizes and networks, due to the overhead of waiting for the processes arriving at the communication phase. Thus, in contrast to HLogGP, a higher prediction error of τ–Lop in this case is due to the difference between the modeled and actual synchronization scenarios rather than the inaccuracy of the model itself.

To further underline the accuracy of the heterogeneous τ–Lop, Fig. 16 compares it to its homogeneous version. In the homogeneous case, the matrices are partitioned into a cluster of 8×8 cores in the Fermi platform. Processes in the same column run in the same node, and hence communicate through shared memory, while they communicate through the network horizontally. Note that even in the case of a far more complex heterogeneous platform, the increase of the prediction error of τ–Lop is quite small with respect to the homogeneous case. The figure also includes the HLogGP heterogeneous model and its counterpart for homogeneous platforms LogGPH [40]. Heterogeneous HLogGP born as an extension of homogeneous LogGPH which manages the additional parameters appearing in a heterogeneous platform. HLogGP becomes LogGPH when the nodes of the platform are identical, and the heterogeneity is only in the communication channels. Finally, Fig. 16 exposes $m\log_n P$ [41], a well-known homogeneous model.

The error of LogGPH and $m\log_n P$ in the homogeneous case is already much higher that that of τ–Lop in the heterogeneous case. It is natural to expect that their extensions to heterogeneous platforms would not behave better. The increase in $\bar{\mu}$ is especially noteworthy in HLogGP with respect to LogGPH in the Infiniband case. The point is that they have fundamental shortcomings in the treatment of the concurrency of the transmissions, as demonstrated in [30]. In TCP, the HLogGP error increasing is not meaningful, despite of its irregular proportional error figures for the range of configurations evaluated in the heterogeneous case.

6 Conclusions

Functional Performance Models describe the computing performance of a process by integrating performance characteristics of both platform and algorithm. They are a useful formalism for computing balancing in heterogeneous platforms. FPMs however lack the ability of accounting for the communication costs in its balancing efforts. Indeed, communication costs have a crucial influence: The FPM driven deployment of the processes on the platform determines the channels used by a process for communicating with the rest. As these channels are usually uneven or highly uneven in terms of capacity, the deployment schedule impacts severely the communication costs. A communication performance model estimates these costs, avoiding an experimental measurement which is always cumbersome, non-portable and expensive. This leads us to envisage that a tight cooperation

of both computation and communication formal models is the way to go to make real progress in the problem of load balancing optimization in heterogeneous environments.

The τ–Lop communication performance model appears to be a suitable tool for this task. Satisfactorily evaluated in multi-core clusters, it is here extended to heterogeneous platforms under some basic and meaningful assumptions. In practical terms, we provide a τ–Lop based software tool which determines the communication cost of a FPM load balanced process deployment avoiding any experimental testing. The SUMMA matrix multiplication and the 2D wave equation solver, representative for many HPC kernels, have been evaluated, though the application to other data parallel kernels is direct. The obtained communication cost predictions show that τ–Lop reaches good accuracy, robustness and scalability in the experimental platform with different network types, Infiniband and Ethernet, and different processing unit types, GPU and multi-core processors. In addition, we contribute with the modeling and cost estimations of the HLogGP model and discuss the reasons of the poor accuracy of its predictions compared to those of τ–Lop.

ACKNOWLEDGMENTS

This publication has emanated from research conducted with the financial support of the Science Foundation Ireland (SFI) under Grant Number 14/IA/2474. This work was partially supported by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS). It was also partially supported by the computing facilities of Extremadura Research Center for Advanced Technologies (CETA-CIEMAT), funded by the European Regional Development Fund (ERDF). We are also in debt to the anonymous reviewers for their insightful comments leading to improvements in this manuscript.

REFERENCES

[1] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, 1970.

[2] B. W. K. S. Lin, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973.

[3] D. F. Gleich, "Hierarchical Directed Spectral Graph Partitioning," Information Networks, Stanford University, Final Project, 2005, 2006, cited over 6 times.

[4] B. Hendrickson, *Graph partitioning and parallel solvers: Has the emperor no clothes?*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1998, pp. 218–225.

[5] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1998.

[6] ——, "Multilevel k-way partitioning scheme for irregular graphs," *J. Parallel Distrib. Comput.*, vol. 48, no. 1, pp. 96–129, Jan. 1998.

[7] H. D. Simon, "Parallel methods on large-scale structural analysis and physics applications partitioning of unstructured problems for parallel processing," *Computing Systems in Engineering*, vol. 2, no. 2, pp. 135–148, 1991.

[8] R. D. Williams, "Performance of dynamic load balancing algorithms for unstructured mesh calculations," *Concurrency: Practice and Experience*, vol. 3, no. 5, pp. 457–481, 1991.

[9] G. Karypis and K. Schloegel. (Version 5.1) METIS: Serial graph partitioning and fill-reducing matrix ordering.

[10] C. Walshaw, M. Cross, and M. G. Everett, "Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes," *J. Parallel Distrib. Comput.*, vol. 47, no. 2, pp. 102–108, 1997.

[11] F. Pellegrini and J. Roman, "SCOTCH: A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs," in *Proc. of the Intnl. Conf. and Exhibition on High-Performance Computing and Networking*, ser. HPCN Europe 1996. London, UK, UK: Springer-Verlag, 1996, pp. 493–498.

[12] F. Pellegrini, *Static Mapping of Process Graphs*. John Wiley & Sons, Inc., 2013, pp. 115–136.

[13] F. Tessier, G. Mercier, and E. Jeannot, "Process placement in multicore clusters:algorithmic issues and practical techniques," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 4, pp. 993–1002, 2014.

[14] J. Dongarra and A. L. Lastovetsky, *High Performance Heterogeneous Computing*. New York, NY, USA: Wiley-Interscience, 2009.

[15] O. Beaumont, V. Boudet, A. Petitet, F. Rastello, and Y. Robert, "A proposal for a heterogeneous cluster ScaLAPACK (dense linear solvers)," *Computers, IEEE Transactions on*, vol. 50, no. 10, pp. 1052–1070, Oct 2001.

[16] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. Whaley, *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, 1997.

[17] A. Lastovetsky and R. Reddy, "Data Partitioning with a Functional Performance Model of Heterogeneous Processors," *Int. J. High Perform. Comput. Appl.*, vol. 21, no. 1, pp. 76–90, Feb. 2007.

[18] D. Clarke, Z. Zhong, V. Rychkov, and A. Lastovetsky, "FuPerMod: A Framework for Optimal Data Partitioning for Parallel Scientific Applications on Dedicated Heterogeneous HPC Platforms," in *12th Int. Conf. on Parallel Computing Technologies (PaCT-2013)*. St. Petersburg, Russia: Lecture Notes in Computer Science 7979, Springer, 30 Sept - 4 Oct 2013, pp. 182–196.

[19] D. Clarke, Z. Zhong, V. Rychkov, and A. Lastovetsky, "FuPerMod: a software tool for the optimization of data-parallel applications on heterogeneous platforms," *The Journal of Supercomputing*, vol. 69, pp. 61–69, 2014.

[20] D. Clarke, A. Lastovetsky, and V. Rychkov, *Column-Based Matrix Partitioning for Parallel Matrix Multiplication on Heterogeneous Processors Based on Functional Performance Models*, ser. Lecture Notes in Computer Science. Springer, 2012, vol. 7155, pp. 450–459.

[21] Z. Zhong, V. Rychkov, and A. Lastovetsky, "Data Partitioning on Heterogeneous Multicore Platforms," in *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, Sept 2011, pp. 580–584.

[22] Z. Zhong, V. Rychkov, and A. Lastovetsky, "Data Partitioning on Heterogeneous Multicore and Multi-

GPU Systems Using Functional Performance Models of Data-Parallel Applications," in *Cluster Computing, 2012 IEEE Int. Conference on*, Sept 2012, pp. 191–199.

[23] J. L. Bosque and L. Pastor, "A parallel computational model for heterogeneous clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 12, pp. 1390–1400, Dec. 2006.

[24] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "LogGP: incorporating long messages into the LogP model - one step closer towards a realistic model for parallel computation," in *Proc. of the seventh annual ACM symposium on Parallel algorithms and architectures*, ser. SPAA '95, NY, USA, 1995, pp. 95–105.

[25] A. Lastovetsky, I.-H. Mkwawa, and M. O'Flynn, "An accurate communication model of a heterogeneous cluster based on a switch-enabled ethernet network," in *Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on*, vol. 2, 2006, pp. 6 pp.–.

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

[27] Y. Ogata, T. Endo, N. Maruyama, and S. Matsuoka, "An efficient, model-based CPU-GPU heterogeneous FFT library," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE Int. Symp. on*, April 2008, pp. 1–10.

[28] S. Y. Chan, T. C. Ling, and E. Aubanel, "Performance Modeling for Hierarchical Graph Partitioning in Heterogeneous Multi-core Environment," *Parallel Comput.*, vol. 46, no. C, pp. 78–97, Jul. 2015.

[29] J.-A. Rico-Gallego and J.-C. Díaz-Martín, "τ–lop: Modeling performance of shared memory MPI," *Parallel Computing*, vol. 46, pp. 14 – 31, 2015.

[30] J.-A. Rico-Gallego, J.-C. Díaz-Martín, and A. L. Lastovetsky, "Extending τ–lop to model concurrent MPI communications in multicore clusters," *Future Generation Computer Systems*, vol. 61, pp. 66 – 82, 2016.

[31] K. W. Cameron, R. Ge, and X. H. Sun, "$\log_m P$ and $\log_3 P$: Accurate Analytical Models of Point-to-Point Communication in Distributed Systems," *Computers IEEE Transactions on*, vol. 56, no. 3, pp. 314–327, 2007.

[32] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda, "High Performance RDMA-based MPI Implementation over InfiniBand," in *Proc. 17th Int. Conf. on Supercomputing*, ser. ICS '03. NY, USA: ACM, 2003, pp. 295–304.

[33] J.-A. Rico-Gallego, J.-C. Díaz-Martín, and A. L. Lastovetsky, "Modeling contention and mapping effects in multi-core clusters," in *Algorithms, Models, and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar)*, ser. Lecture Notes in computer Science. Springer International Publishing, 2015.

[34] R. A. van de Geijn and J. Watts, "SUMMA: Scalable Universal Matrix Multiplication Algorithm," Austin, TX, USA, Tech. Rep., 1995.

[35] K. Hasanov, J.-N. Quintin, and A. Lastovetsky, "Hierarchical Approach to Optimization of Parallel Matrix Multiplication on Large-Scale Platforms," *The Journal of Supercomputing*, vol. 71, pp. 3991–4014, 11/2015 2015.

[36] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix-matrix multiplication on heterogeneous platforms," in *Parallel Processing, 2000. Proceedings. 2000 International Conference on*, 2000, pp. 289–298.

[37] T. Malik, V. Rychkov, and A. Lastovetsky, "Network-aware optimization of communications for parallel matrix multiplication on hierarchical hpc platforms," *Concurrency and Computation: Practice and Experience*, vol. 28, pp. 802–821, 03/2016 2016.

[38] J.-A. Rico-Gallego and J.-C. Díaz-Martín. (2016, August). [Online]. Available: http://gim.unex.es/taulop

[39] Z. Zhong, V. Rychkov, and A. Lastovetsky, "Data partitioning on multicore and multi-gpu platforms using functional performance models," *IEEE Transactions on Computers*, vol. 64, pp. 2506–2518, 09/2015 2015.

[40] L. Yuan, Y. Zhang, Y. Tang, L. Rao, and X. Sun, "Loggph: A parallel computational model with hierarchical communication awareness," in *Computational Science and Engineering (CSE), 2010 IEEE 13th International Conference on*, dec. 2010, pp. 268 –274.

[41] B. Tu, J. Fan, J. Zhan, and X. Zhao, "Performance analysis and optimization of mpi collective operations on multi-core clusters," *The Journal of Supercomputing*, vol. 60, no. 1, pp. 141–162, 2012.

Juan-Antonio Rico-Gallego received the Computer Science Engineering degree and the PhD degree on Computer Science from the University of Extremadura in 2002 and 2016 respectively. He is currently Associate Professor in the Department of Computer Systems Engineering of the University of Extremadura (Spain). His research interests are in MPI implementations and applications and in performance models on heterogeneous platforms. Other research interests include current trends and issues concerning Exascale software. He codevelops AzequiaMPI, a thread-based MPI 1.3 standard implementation.

Alexey L. Lastovetsky received a Ph.D. degree from the Moscow Aviation Institute in 1986, and a Doctor of Science degree from the Russian Academy of Sciences in 1997. His main research interests include algorithms, models, and programming tools for high performance heterogeneous computing. He is currently Associate Professor in the School of Computer Science at University College Dublin (UCD). At UCD, he is also the founding Director of the Heterogeneous Computing Laboratory.

Juan C. Díaz-Martín received the Computer Science degree and the Ph.D. in Computer Science from the Polytechnic University of Madrid, in 1988 and 1993 respectively. He is currently Associate Professor in the Department of Computer and Communication Technology of University of Extremadura (Spain). His research interest is in MPI implementation. He codevelops and maintains AzequiaMPI, a thread-based MPI 1.3 standard implementation that has been ported to networks of DSPs and soft-core processors.