UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Máster Universitario en Ingeniería Informática

Trabajo Fin de Máster

Equilibrado de Carga Dirigido por Modelos de Kernels de Datos Paralelos en Plataformas Heterogéneas de Alto Rendimiento

Sergio Moreno Álvarez

Enero, 2019

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Máster Universitario en Ingeniería Informática

Trabajo Fin de Máster

Equilibrado de Carga Dirigido por Modelos de Kernels de Datos Paralelos en Plataformas Heterogéneas de Alto Rendimiento

Autor: Sergio Moreno Álvarez

Tutor: Juan Antonio Rico Gallego

Co-Tutor: Juan Carlos Díaz Martín

Tribunal Calificador

Presidente: Antonio Plaza Miguel

Secretario: José Mª Granado Criado

Vocal: Félix Rodríguez Rodríguez

INDEX OF CONTENTS

INDEX OF TABLES

INDEX OF FIGURES

RESUMEN

Resumen —

Las aplicaciones de datos paralelos se componen de varios procesos que aplican el mismo cómputo (kernel) a diferentes conjuntos de datos. Además, durante su ejecución, estas aplicaciones necesitan comunicar resultados parciales. Las plataformas heterogéneas son aquellas donde cada recurso de cómputo del sistema es probablemente diferente a los otros, y están compuestas por aceleradores. La conexión entre los elementos se realiza mediante redes de diferente rendimiento y características. Estos tienen que trabajar juntos para ejecutar una aplicación o resolver un problema, lo cual es lo complicado de este escenario. Por ello, el problema del equilibrado de carga de las aplicaciones paralelas de datos en plataformas heterogéneas se está investigando y resolviendo mediante distribuciones no uniformes de la carga de trabajo entre todos los recursos disponibles. Este problema se ha demostrado NP-Completo. La literatura ha desarrollado varias heurísticas para encontrar soluciones óptimas en las que diferentes modelos de rendimiento de computación y comunicación se utilizan como métrica en los algoritmos de partición. Los modelos nos permiten describir el funcionamiento del sistema, mientras que las heurísticas son el enfoque que se utiliza para encontrar una solución satisfactoria. Discutimos el papel de estos modelos y, finalmente para mejorar estos enfoques heurísticos, sustituimos métricas basadas en volumen de comunicaciones por una métrica basada en los tiempos de comunicaciones. Estos tiempos son obtenidos mediante un modelo

analítico a través de una herramienta simbólica que manipula, evalúa y representa el coste de la comunicación de una partición con una expresión analítica utilizando el modelo de rendimiento de comunicación τ–Lop.

Palabras clave — Modelado de Rendimiento en Comunicaciones, Modelos Funcionales de Rendimiento de Cómputo, Plataformas Heterogéneas, Algoritmos de Particionamiento, Optimización de la Comunicación, Kernels de Datos Paralelos.

ABSTRACT

Abstract — Data-Parallel applications are composed of several processes that apply the same computation (kernel) to different amounts of data. While its execution, these applications need to communicate partial results. The heterogeneous platforms are those where each computation resource of the system is probably different from the others, and are composed of accelerators. The connection between the elements is made through networks of different performance and characteristics. These have to work together to execute an application or solve a problem, which is the complicated part of this scenario. Therefore, the load balancing problem of Data-Parallel applications in heterogeneous platforms is being investigated and solved by non-uniform distributions of the workload among all available resources. The objective of this solution is to find a partition that minimizes the cost of computation and communication, which is not trivial. This problem is demonstrated as NP-Complete. The literature has developed several heuristics to find optimal solutions where computation and communication performance models are used as metrics in the partitioning algorithms. The models allow us to describe the functioning of the system, while heuristics are the approach used to find a satisfactory solution. We discuss the role of these models and finally, to improve these heuristic approaches, we replace metrics based on communications volume with a metric based on communication times. These times are obtained through a symbolic tool that manipulates, evaluates and represents the cost of communication of a partition with an analytic expression using the

communication performance model τ–Lop.

Keywords — Communication Performance Modeling, Functional Computation Performance Models, Heterogeneous Platforms, Partitioning Algorithms, Communication Optimization, Data-Parallel Kernels.

1

INTRODUCTION & MOTIVATION

There are some terms we should know to understand the scope of work. Data-Parallelism applies computation to every single component of a data collection, so the workload is directly proportional to the full size of data and dependent on the data locality. This paradigm allows parallelism scaling with the amount of data. Data-Parallel Kernels or kernels, are the main components of scientific applications running on High Performance Computing Platforms (HPC). In heterogeneous platforms, the kernel workload or data space is non-uniformly distributed between all the processes in proportion to their features, specifically in their computational capabilities (speed). With this, we can avoid the waiting times of the faster processes with respect to the slow ones and obtain a balanced workload. A key point here is to determine accurately the speed of a process. There are two techniques used to represent the speed with a numerical value or with a more elaborated function of the workload size. These techniques are Constant Performance Models (CPM) and Functional Performance Models

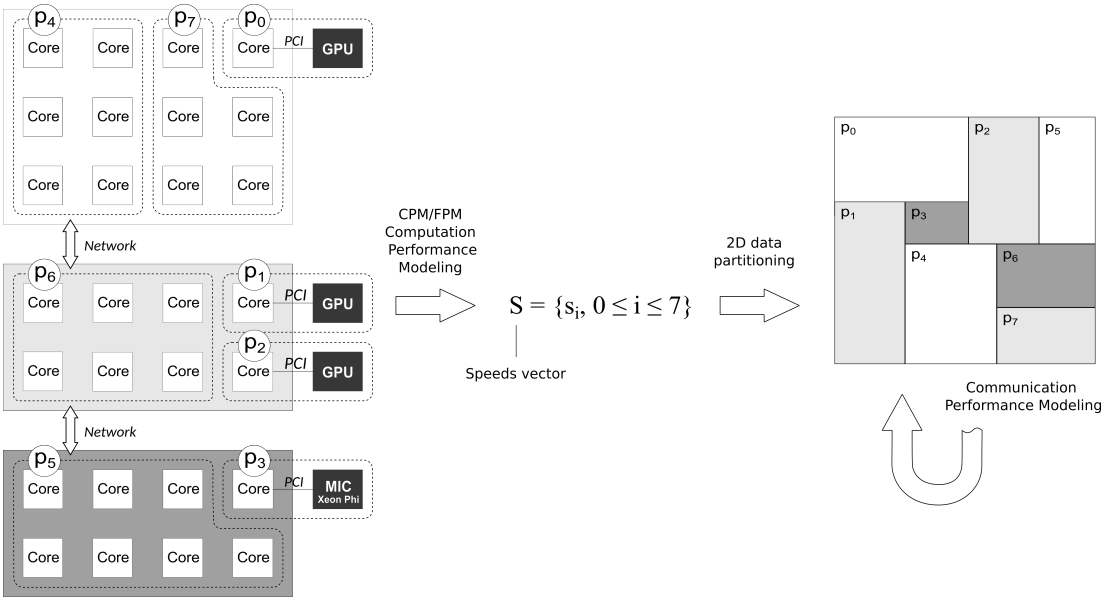


Figure 1.1: Communication channels, as shared memory and network, often show far different performance. Note that neighbor data regions (probably with higher interaction) have been assigned to processors linked by the slower network channel. The overall result is that such partition introduces so significant communication inefficiencies that may completely defeat the original balancing purpose. A Communication Performance Model of the kernel should guide the elaboration of load-balancing data-partitions of optimal communication costs. Processes are represented as p_i.

(FPM) respectively. The assigned area to every processor is directly proportional to its speed. Lastovetsky et al., 2007 define $s_i(x)$ as a function of the problem data size x. To define the FPMs of the processors, a speed array is used representing these speeds to their corresponding processor. With this, a certain partition is created as we can see it in *Figure 1.1*, which is made up of rectangles assigned to a specific processor. Each rectangle will use its computational resources to process the amount of data given to it. These rectangles are characterized by their position in the matrix by starting coordinates and dimensions. Therefore, given a problem to be solved, the goal is to determine the shape of the rectangles and their position in the partition to minimize the execution time of the kernel.

All the processors execute the same program, but applied to a different data partition. The code typically executes a loop. Each step of the loop has two phases, the communication with the rest of processes, and the computation of

its own data in the rectangle. However, leaving aside the seminal mathematical work of Beaumont et al., 2001, current practice on partitioning algorithms have disregarded communication costs. The communication may have a determinant influence in the overall execution time of a kernel. Therefore, a partition with a balanced load in terms of computing can lead to this same partition having a communication that affects the total time of execution of the kernel, and in this way, the computational load balance process is useless. In this work we introduce accurate predictions of the communication cost of a numerical kernel on a given partition using an *Analytical Performance Communication Model*, with the aim of using it in partitioning and optimization strategies.

The complexity of numerical kernels has grown in tandem with the complexity of the underlying computing platforms. Accelerated computing, mostly known nowadays as Heterogeneous Computing has quickly changed the High Performance Computing field. Current HPC platforms are composed of multi-core nodes and accelerators connected by networks of different performance and features. Graphics processing units (GPUs) are increasing the energy efficiency of a computing system by offering a good performance per watt ratio. As a result, the usage of a GPU to accelerate an application has been a standard approach for several years.

The motivation of this work is the evaluation of the cost of communications as a metric for data partitioning in heterogeneous platforms to achieve an improvement in the overall execution time and performance of a kernel. For this we have modeled the communication based on specific parameters of the platform where the kernels will be executed, knowing this way, the characteristics and properties which defines the platform, an issue that will be explained later. We use an analytical expression to predict the cost of communication. This expression is applicable to any type of communication, either the estimation of a complete kernel, a collective or a simple point-to-point communication. It is quantified in terms of units of time. Until now, this communication metric has not been modeled on heterogeneous platforms due to its lack of precision. We

focus on the τ–Lop analytical communication performance model Rico-Gallego et al., 2015; Rico-Gallego et al., 2017, because it considers contention and process mapping, has a rich expressiveness and has demonstrated good accuracy on both homogeneous and heterogeneous platforms.

2

OBJECTIVES

This chapter presents the objectives of the Master Thesis. Studying certain partitioning algorithms, we have detected possible improvements to introduce in this scope. Therefore, using the τ–Lop library proposed by Rico-Gallego et al., 2015; Rico-Gallego et al., 2017, we perform its deployment as a metric in this type of partitioning algorithm. In addition, during the development of this work, we find the need to perform a comprehensive measurement of the parameters that make up this model, so, we explain the process performed in detail in order to demonstrate the effectiveness of this measurement. Finally, we propose a hierarchical heuristic that aims to improve others proposed by other authors in their respective works. We will study an example of these proposals (Beaumont et al., 2001), and we will improve their contribution. Thus, main contributions of this Master Thesis are:

1. A new communication-based metric for data partitioning in heterogeneous

platforms, based on the τ–Lop analytical performance model.

2. Prove flexibility and accuracy of the new approach for introducing new heuristics to the partitioning algorithms based on new locality and hierarchical orders.

Also, we can define specific secondary objectives during this document:

1. Carry out the measurement of the model parameters in different communication patterns and channels for a specific platform.

2. Analyze how the τ–Lop model adapts to the different kernels on HPC platforms.

During this work we used a tool that mechanizes the construction and evaluation of τ–Lop communication cost expressions, in particular those of hybrid kernels which provides with a C++ procedural interface to represent and evaluate any τ–Lop cost expression.

3

A BRIEF REVIEW OF COMMUNICATION PERFORMANCE MODELS

An analytical communication performance model represents communications as a parameterized formal expression which is evaluated to obtain the cost of the communication. These expressions are represented in terms on time. Communication performance models are characterized by a set of parameters which are different for each platform due to its specific features. Firsts models were proposed more than two decades ago for the clusters of that time, composed by single processor nodes. *LogP* Culler et al., 1993, a foundational model for the then-emerging homogeneous clusters represents the cost of a communication by four parameters: L is the *network delay*, and represent the latency of the network, o is the *overhead* or cycles that a CPU devotes to send the message, g is the *gap per message* and represents the minimum time interval between two

consecutive injections to the network, and, finally, P is the number of processes. *LogP* model was improved by *LogGP* Alexandrov et al., 1995, which includes a new parameter G (gap per message) allowing to represent the influence of the network bandwidth in the transmission of large messages. In *LogGP*, shown in Figure 3.1, the cost of a *point-to-point transmission* of a message of size m is represented as: $T_{p2p}(m) = 2\,o + L + (m-1)\,G$. More advanced models have been later proposed, as *PlogP* Kielmann et al., 2000, that considers parameters gap per message and overhead linear functions of the message size, achieving higher accuracy.

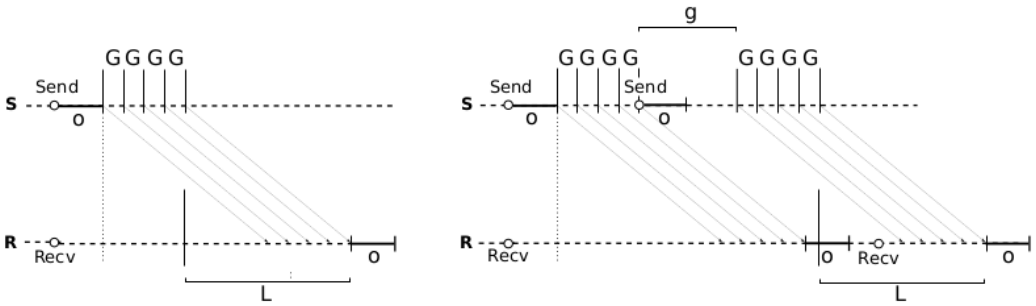


Figure 3.1: Representation of the cost of a point-to-point message transmission under the LogGP model, one message of size $m = 5$ at the left and two messages at the right. Note that the gap delays the emission of the second message

Unlike the homogeneous case, modeling the cost of the communications of current heterogeneous platforms is so far a relatively unexplored field, and nonetheless a central one. Heterogeneous models proposals are the *HLogGP* model Bosque et al., 2006, based on *LogGP*, the *LMO* model Lastovetsky et al., 2006, based on the *Hockney* model, and lately τ–Lop Rico-Gallego et al., 2017. The scalar parameters of *LogGP* are expanded in *HLogGP* to represent the o_s, o_r and g as vectors of P components, where P is the number of processors in the machine. L and G now depend on each pair of processors in the network, so they become matrices of $P \times P$ components. A problem of this model is that the parameters have to be measured for each pair of processors in the system, and hence, the number of tests is of order $O(P^2)$. The cost expression of a point-to-point message transmission between processors p_i and

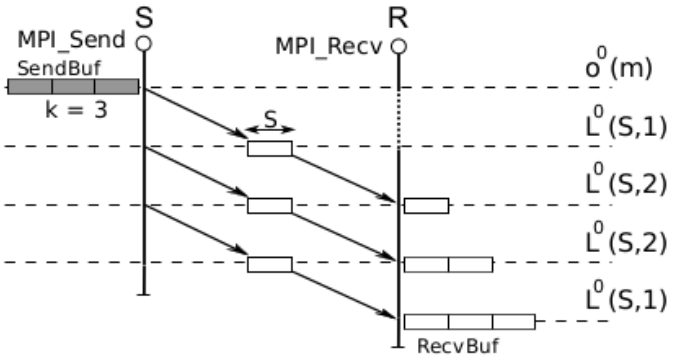


Figure 3.2: τ-Lop p2p cost analysis $T^0_{p2p}(m)$ of a shared memory transmission of a message split up in $k = 3$ segments of size S. The transmission entails four steps and six transfers (arrows).

p_j can be represented as $T_{i\to j}(m) = L_{ij} + o_{S_i} + o_{R_j} + (m-1)\,G_{ij}$. LMO model targets the impact of the heterogeneity of the processors on the communication cost of a set of operations, namely point-to-point, one-to-many (scatter and gather) and broadcasting. The cost of a message transmission is defined as $T_{i\to j}(m) = L_{ij} + C_i + m\,t_i + C_j + m\,t_j + \frac{m}{\beta_{ij}}$, where C_i is the *fixed processing delay* of process p_i, t_i is the *per-byte delay*, $L_{i\,j}$ is the *fixed network delay*, and β_{ij} is the transmission rate of the channel connecting the processors p_i and p_j. Like *HLogGP*, the number of parameters is of order $O(P^2)$ in a generalized P-node cluster. τ–Lop, shown in 3.2, is an analytical communication performance model that considers contention and process mapping with a rich expressiveness and has demonstrated good accuracy on both homogeneous and heterogeneous platforms Rico-Gallego et al., 2017. τ–Lop will be explained later in Section 5.1.

4

PROBLEM DEFINITION

We studied a load-balanced partition of a 2D data-space between a set of processes characterized by different computing power based on Beaumont et al., 2001 as a formal optimization problem. They also proposed a heuristic with its variables, constraints and objective function based on *dynamic programming* which builds a column-based partition. The solution of Beaumont et al., 2001 is explained in Section 6.1.

Collective operations is a concept in parallel computing in which data is simultaneously sent to or received from many processes. Models described in Chapter 3 lead to a poor expressive power on collective operations. They ignore the network topology and the contention in the network channels. As a result, they are not able to accurately capture the cost of the complex patterns which arise in the communication phases of data-parallel kernels. The growing trend to heterogeneous computing poses hence substantial challenges for modeling their communications:

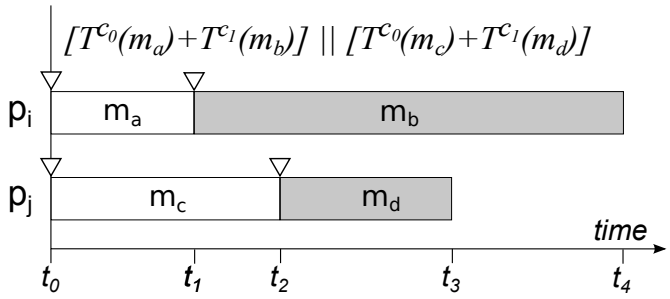


Figure 4.1: Cost modeling scenario commonly found in heterogeneous platforms. Processes p_i and p_j transmit a sequence of two messages of different size through two different communication channels c_0, in white, and c_1, in grey. The processes compete for c_0 from t_0 to t_1. Later for c_1 from t_2 to t_3. Above is the overall τ–Lop cost, an expression hard to evaluate in practice.

1. Accuracy: The prediction of the communication times of a certain partition in an accurate way is very important due to the high impact that this can have on the total execution time in the current heterogeneous platforms. To do this, we use τ–Lop communication performance model and we demonstrate its accuracy with respect to real communication times.

2. Contention awareness: This is a very important aspect to take into account as it significantly influences communication times, and is also one of the main contributions of the τ–Lop model. Transmission are represented in Figure 4.1, that are made by the processes by sending a message through a channel. Contention represents the number of concurrent transmissions through the same channel, which negatively influences the time it takes for such transmission to finalize, since they all use the same channel.

3. Process mapping awareness: Process mapping directly influences communication cost. This can be seen in the Figure 4.2, where rectangles are made by the amount of data given to a process in a specific node. This is because the organization of the rectangles assigned to each process in a partition can carry slow communications due to the fact that most communications are made through slow channels. Nodes are represented with different colors in Figure 4.2, what causes that communication between different nodes are through network channel, and shared memory between processes on the

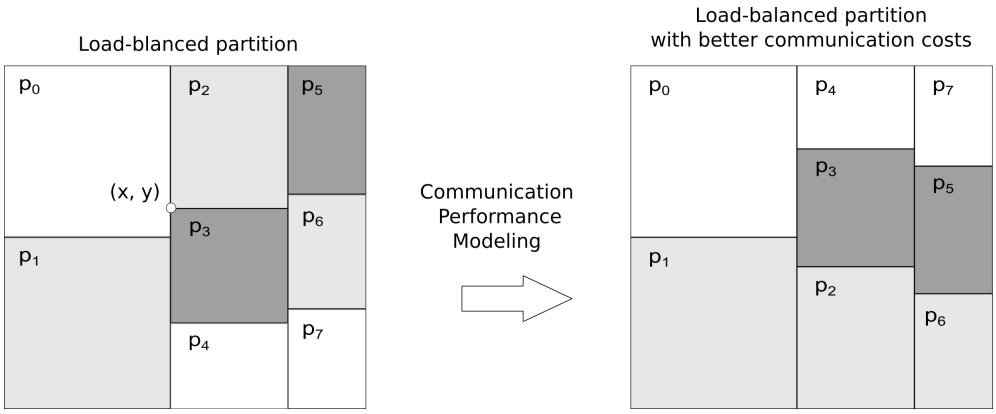


Figure 4.2: A load-balanced partition can be altered afterward using a Communication Performance Model. Note that the second tiling should be more efficient in terms of communication costs.

same node.

4. Manageable cost expressions: The transmissions are represented as expressions in τ–Lop model as we can see in Figure 4.1. These expressions in heterogeneous platforms become unmanageable. Thus, we need tools for supporting the automatic generation, composition and evaluation of cost expressions. We used a tool named τ–Lop Tool in Section 5.2 which resolve this situation.

Beaumont et al., 2001 calculate the communication time using the total volume of data moved between processes. It represents the total volume of communications, and is minimized as the rectangles become as square as possible. The solution minimizes the volume of communication using as a metric the sum of the half perimeters of the resultant rectangles. They discussed the matrix multiplication of dense matrices using the SUMMA algorithm, described in Section 5.2.3.

The rational behind this metric is that communication volume is proportional to the perimeter of the resultant rectangles, which is minimized when rectangles become as square as possible. They solve this optimization problem of the communication volume with a integer program, which has three inputs: P number of processes, size of the 2D data space (N^2), and the speed vector

$S = \{s_1, s_2, \ldots, s_P\}$, where s_i is the relative speed of the process p_i. The objective is to tile the data space using the proportional speed of the processes with P non-overlapping rectangles of size $n_i = w_i \times h_i, 1 \leq i \leq P$. The program has various restrictions. The two first restrictions (4.1a) and (4.1b) groups the whole data space using the P non-overlapping rectangles for a good tiling. Last restriction (5.1a) is used to balance the load of a single partition in a non-uniform way, where every process is related with a rectangle of an area proportional to its speed. Thus, every process spends the same $t_i = n_i/s_i$ computational time in solving its assigned rectangle, or at least, little differentiated computation times $t_i = n_i/s_i$. The goal is to find a partition of the data space which minimizes the communication cost.

$$\sum_{i=1}^{P} n_i = 1. \tag{4.1a}$$

Rectangles are non-overlapping and tile the unit square. (4.1b)

$$\frac{n_1}{s_1} = \frac{n_2}{s_2} = \cdots = \frac{n_P}{s_P}. \tag{4.1c}$$

The total volume of Beaumont communication metric β disregards the real time cost of communications. Facing the challenges above discussed requires a more sophisticated metric capturing the characteristics of the heterogeneous platform and the specific kernel communication patterns. We used τ–Lop communication performance model proposed by Rico-Gallego et al., 2017 as an alternative metric, an approach that should be able to output a more accurate communication cost estimation. As pointed out in Chapter 3, a communication performance model represents communications as a parameterized formal expressions. The evaluation of this expression determines the cost of the communication in terms of real time.

Once the problem is defined, in Chapter 5 we will study the τ–Lop Model to know it in depth and how to apply it as a metric to the Beaumont problem described in Section 6.1. Also, the measurements of the parameters for the model included in Section 5.3 is a contribution of this Master Thesis. The

implementation and explanation of this metric is described in Section 6.2.1, and later we propose an alternative heuristic for data partitioning in Section 6.2.2.

5

τ-LOP MODEL STUDY

Rather than the half-perimeter expression used for Beaumont algorithm β, the new optimization objective is finding the partition which minimizes its τ–Lop cost expression Θ, explained next. Thus, the full partition space generated by Beaumont et al., 2001 algorithm is now reevaluated tagging each tiling with its Θ value, superseding the total volume of communications metric β previously used.

To implement a metric based on τ–Lop, we must first know how this library works. We will study that in this section, where we will learn the use of τ–Lop and later use it as a metric in the implementation.

5.1 Model Description and Explanation

The τ–Lop communication performance model represents a point-to-point message *transmission* as a sequence of data *transfers*. It can model following

features:

1. Concurrent transfers: the channel bandwidth shrinks if the transfers are transmitted at the same channel and time. This causes the memory bandwidth gets exhausted. As a result, the cost grows linearly with the number of cores.

2. Message segmentation: demands less intermediate memory because storing the whole message is not needed. In addition, it speeds up the communication progress by overlapping the sending of segments with their reception.

3. Collectives: which implements the possibility to have multiples senders and receivers simultaneously.

4. Protocol cost additions: limitations on buffer space impose MPI libraries to set up communication protocols, such as rendezvous, so that sender waits for receiver, who on arrival notifies the sender to proceed. Protocols charge additional cost, mainly at the beginning of the communication.

τ–Lop is constituted by different parameters that explain below. $L^c(m, \tau)$, known as the *transfer time*, is the cost of τ concurrent transfers of size m through a communication channel c. $o^c(m)$, known as the *message overhead*, represents the time to start the data injection into the channel. The different values $o^c(m)$ and $L^c(m, \tau)$ are known as the *model parameters*, specific for each target platform. The number of transfers composing a transmission depends on the nature of the used channels and the communication middleware. The shared memory channel ($c = 0$), for instance, uses an intermediate memory buffer between source and destination user buffers, so a transmission in shared memory needs two transfers, with cost $T^0_{ptp}(m) = o^0(m) + 2\,L^0(m, 1)$. This can be appreciated in *Figure 5.1*. In a network channel ($c = 1$) as Infiniband, communication libraries usually take advantage of its RDMA capabilities to transmit a message in only one transfer, hence with a cost $T^1_{ptp}(m) = o^1(m) + L^1(m, 1)$. As a transfer may flow with others

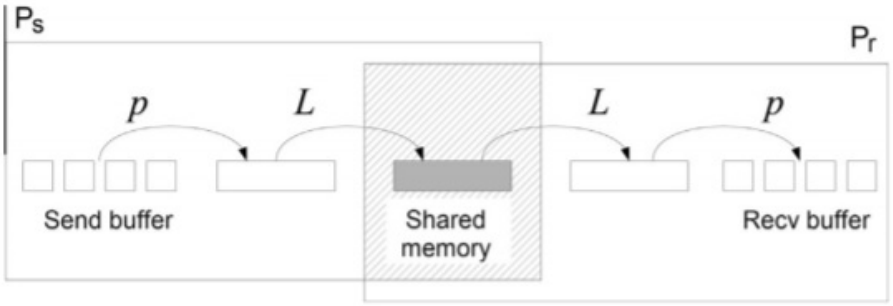


Figure 5.1: A message transmission in τ–Lop is composed by two transfers on the shared memory channel. p is the packing time which happen when the message is not contiguous in memory, and L is the transfer time.

competing for the bandwidth of channel c, its cost $L^c(m, \tau)$ depends not only on the message size m, but also on the number τ of such *concurrent transfers*.

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. τ–Lop represents the cost of A concurrent transfers with the $\|$ operator, defined 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 to the transmission level so that $A \,\|\, T^c(m)$ represents the cost of A *concurrent transmissions* of a message of size m contending for the channel c.

$$\mathrm{T}^0_{p2p}(m) = o^0(m) + 2L^0(m, 1). \tag{5.1a}$$

This can be appreciated in *Figure 5.2* where two point-to-point transmissions concurrently progressing through the shared memory channel. Each transmission, with the cost defined by (5.1a), is composed of two transfers actually contending for the communication channel, leading to an increase in the cost:

$$2 \,\|\, \mathrm{T}^0_{p2p}(m) = 2 \,\|\, [o^0(m) + 2L^0(m, 1)] = o^0(m) + 2L^0(m, 2). \tag{5.2a}$$

The concurrency of transfers has its impact on the cost of transmissions. The expression A$\|$ $L(m,\tau)$ represents the cost of A concurrent sets of in turn τ

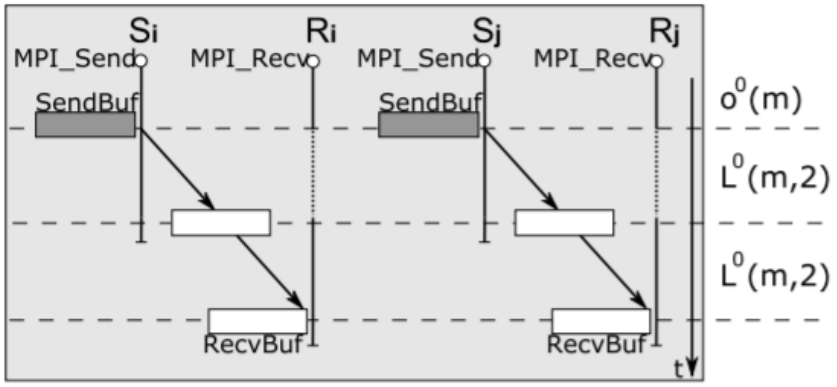


Figure 5.2: Cost of the concurrency of two transmissions in shared memory through a intermediate buffer expressed in terms of τ–Lop.

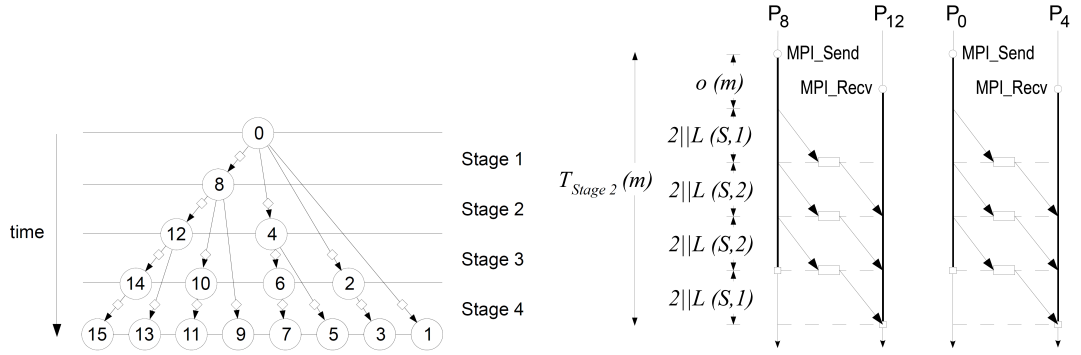


Figure 5.3: τ–Lop cost of stage 2 of a binomial broadcast, expressed in terms of concurrent transfers.

concurrent transfers. It is defined as A$\| L(m,\tau)$ = L(m,A$\times\tau$).

As example of concurrent transmission in a homogeneous platform, the binomial tree broadcast algorithm is used for instance for short messages and a small number of processes. The number of involved processes grows with successive stages, while the message size remains constant. The height of a binomial tree of P processes is $h(P) = [log_2(P)]$, and hence it requires $[log_2(P)]$ stages.

Figure 5.3 illustrates the concept. It shows the pair of message transmissions A=2 taking place at second stage of the binomial tree broadcast. One of these transmission is the process p=8 sending to process p=12, and the next

transmission is the process p=0 sending to process p=4. Both transmission progress concurrently through shared memory. The cost of the first stage equals the cost of a message transmission T(m). The cost of subsequent stages is expressed by next equation as $2 \| T(m), 4 \| T(m), 8 \| T(m)$:

$$\mathrm{A} \| L(m, \tau) = A \| \sum_{j=0}^{s-1} L_j(m, \tau_j) \tag{5.3a}$$

Hence, the cost of the whole broadcast on a full tree would be as follows:

$$\Theta_{bin}(m) = \sum_{i=0}^{[log_2(P)]-1} (2^i \| T(m)) \tag{5.4}$$

Homogeneous platforms cost expressions are formulated as $n \| T^c(m)$ and $T^c(m_1) + T^c(m_2)$. However, in heterogeneous platforms expressions as $T^{c_1}(m_1) \| T^{c_2}(m_2)$, are common. Rico-Gallego et al., 2017 extends τ–Lop with three simplifying formal assumptions that allow us to evaluate these more complex cost expressions. They are needed to unblock the expansion path of these expressions on L terms, while still keeping a high overall accuracy in real hybrid kernels.

A1. A sequence of transmissions progressing through the same channel has the cost of a single transmission of a message of aggregate size, that is, $A \| T^c(m_1) + A \| T^c(m_2) = A \| T^c(m_1 + m_2)$.

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

With these expressions, τ–Lop allows us to address the problems related to

containment and model concurrent transmissions, making it a simple task. As an example, we take the expression from *Figure 4.1*.

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

It leads to different developments depending on the relative message sizes. Thus, in *Figure 4.1*, at time t_0 we have two concurrent transmissions of messages m_a and m_c through the same channel c_0, where $m_c \geq m_a$, $m_b \geq m_d$, and $T^{c_1}(m_b) \geq T^{c_0}(m_c - m_a)$. Applying $A2$, the expression becomes:

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

This expression is made up of two terms. The first one, $2 \parallel T^{c_0}(m_a)$ represents the cost of time between t_0 and t_1. The second one represents the cost of two transmissions that start at t_1 and progress through different channels. Therefore, these two transmissions are represented as individual costs $T^{c_1}(m_b)$ and $T^{c_0}(m_c - m_a)$. When applying the third axiom $A3$ 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, $T^{c_0}(m_c - m_a)$ first term results in $2 \parallel T^{c_0}(m_a) + [T^{c_1}(m_b) \parallel T^{c_1}(m_d)]$, where we have two transmissions that progress concurrently through c_1. By applying $A2$ to the second term, the cost finally becomes an expression directly evaluable using the parameter values of the model:

$$2 \parallel T^{c_0}(m_a) + 2 \parallel T^{c_1}(m_d) + T^{c_1}(m_b - m_d),$$

As we have seen, τ–Lop allows us to estimate the cost of communications on heterogeneous platforms. All this with a precision shown in Rico-Gallego et al., 2017. Even so, the process is quite complex, which is a practical problem to use the library. Because of this, it is necessary to create a tool that automates the process of composing, storing and evaluating cost expressions, whether for homogeneous or heterogeneous systems. Next, we will explain the operation of the tool for the automation of this process.

Algorithm 1 Creation of simple transmissions

Transmission *T_x = new Transmission (c, m, τ);
Process *src = new Process (src_rank, src_node);
Process *dst = new Process (dst_rank, dst_node);
Transmission *T_y = new Transmission (src, dst, m, τ);

5.2 τ–Lop Tool for Composing Cost Expressions

As we have seen before in *Figure 4.1*, cost expressions from τ–Lop are represented as a *set of sequences of transmissions which progress concurrently* for modeling communications with the following tool. In this section, we describe the tool interface and how to use it. There are three main steps to obtain the communication cost. First, we must describe individual message transmissions. Then, we can build concurrent sequences of transmissions. Finally, τ–Lop library evaluates the communication cost of the modeled problem.

5.2.1 Modeling Simple Transmissions

The fundamental software object of the τ–Lop Library is the *Transmission*. Its three attributes c, m and τ represent the cost of an individual τ–Lop transmission of the form $\tau \parallel T^c(m)$. Also, there are others objects. *Process* object, it contains the node where it runs with its rank in the kernel. Algorithm 1 shows two ways of creating a Transmission, specifying the channel in the case of instance T_x, and by specifying the source and destination processes, as in the case of T_y, where the channel is internally determined. These channels are integer numbers starting from 0, assigned increasingly to the highest performance communication channel. For example, $c = 0$ could be shared memory, $c = 1$ IB-network channel and $c = 2$ TCP-network channel.

To offer the principle of composition, τ–Lop provides two higher level objects. *TauLopSequence* contains one or more transmissions carried out in sequence. *TauLopConcurrent* contains a set of *TauLopSequence* objects that progress concurrently. We can see this in the Algorithm. We create simple cost expressions.

Object $conc_1$, is a sequence of two simple transmissions that propagate through a channel c of the form $T^c(m_1) + T^c(m_2)$, and then $conc_2$, two concurrent transmissions on the same channel c of the form $T^c(m_1) \parallel T^c(m_2)$. With this we can evaluate the heterogeneity of the different platforms that are composed of multiple transmissions through different channels. In this respect Algorithm 2 composes the expression of *Figure 4.1* and evaluates it through the *TauLopCost* object.

Algorithm 2 Cost expressions for sequences and concurrents transmissions.

```
TauLopSequence *seq1 = new TauLopSequence ();
seq1→add (new Transmission(c, m1, 1));
seq1→add (new Transmission(c, m2, 1));

TauLopConcurrent *conc2 = new TauLopConcurrent ();
TauLopSequence *seq2;
seq2 = new TauLopSequence ();
seq2→add (new Transmission (c, m1, 1));
conc2→add (seq2);
seq2 = new TauLopSequence ();
seq2→add (new Transmission (c, m2, 1));
conc2→add (seq2);
```

5.2.2 Modeling Collective Operations

This tool allows modeling and estimating the cost of MPI-like collective operations with different algorithms. A collective executes in the context of a *communicator*, a central concept to MPI. With this tool, using a function we cat set the communicator with the rank-to-node map, which defines the communication channel used to communicate every two ranks. There are two types of mapping used for MPI jobs, they are known as *MAPPING_SEQ* and *MAPPING_RR*, which are Sequential and Round Robin respectively. The rank numbers are sequentially assigned to processing units on the same node up to complete the node, then the assignment continues with the next node, and so on. In *MAPPING_RR*, which is the default option in MPICH, rank numbers are sequentially assigned jumping over the nodes of the cluster. There are also other

Algorithm 3 Estimate of the communication cost of a broadcast collective.

```
Communicator *world = new Communicator (P);
Mapping *seqMap = new Mapping(P, Q, MAPPING_SEQ);
world→map(seqMap);
int root = 2;
int m = 1024;
Collective *bcast = new BcastBinomial();
double t = bcast→evaluate(world, &m, root) ;
```

kinds of mapping which are more irregular.

Algorithm 3 predicts the cost of a binomial tree *broadcast* between P processes deployed in sequential mapping on a homogeneous multi-core machine with P/Q nodes, being Q the number of cores per node. It is necessary to clarify that m is hence a vector containing the message size of each involved process, indexed by process rank. In the broadcast case, only the *root* process needs to specify the size of the message, what it does in the position 0. That is why m is passed by reference to *evaluate*.

However, the cost of collective operations depends on many factors. Among these factors is the number of processes, the characteristics of the network, the message size, the type of algorithm used and its implementation. Some of these factors can be observed in the Algorithm 4. This algorithm predicts the cost of a collective implemented by two different algorithms, called *Ring* and *Recursive Doubling* (RDA). *Ring* executes the algorithm in $P - 1$ steps, where in each of these the process with rank p sends a message of size m to the process with rank $p + 1$ and receives it from $p - 1$. *RDA* executes the algorithm in $\log_2 P$ steps, doubling the size of the message in each of its steps, where each process p communicates with the process $p \oplus 2^s$ in each step. In the *Allgather* algorithm, each process sends a message m and receives a message $P \times m$. Therefore, the impact of different mapping algorithms and message sizes m on the cost of communication can be directly evaluated. The tool provides an interface to implement additional collectives and their algorithms.

For instance, a simple *Binomial Tree* algorithm has a cost under τ–Lop given

Algorithm 4 Comparing the communication cost of two algorithms executing the Allgather collective.

```
int P = 8;
int nodes = {0, 1, 1, 2, 0, 2, 1, 0};
int m = {...} ;
Communicator *comm = new Communicator (P);
Mapping *custMap = new Mapping(P, nodes);
comm→map(custMap);

Collective *allg_ring = new AllgatherRing();
Collective *allg_rda = new AllgatherRDA();

double t_ring = allg_ring→evaluate(comm, &m);
double t_rda = allg_rda→evaluate(comm, &m);
```

by $\Theta_{Bin}(m) = \sum_{i=0}^{\log_2(P-1)} [2^i \| T^c(m)]$, capturing the fact that each stage doubles the number of involved processes. The τ–Lop capability of modeling contention is key in the accurate cost modeling current kernels.

5.2.3 Modeling Data-Parallel Kernels

The formal expression Θ produced by modeling a kernel rapidly becomes complex enough to require an automatic processing. It is necessary to use a tool for supporting the automatic generation, composition and evaluation of cost expressions, in particular that of a data-parallel kernel. τ–Lop tool, a C++ procedural interface to represent and next evaluate any cost expression which targets real world data-parallel scientific applications, such as linear algebra packages, digital signal processing, computational fluid dynamics, etc.

In this section, we model three of two kernels which show different features, communication patterns. The first kernel, Wave2D, solves a partial differential wave equation in 2D using a 1D data partition. It uses point-to-point communication between the processes involved in the algorithm. The second kernel is a distributed matrix multiplication algorithm, named SUMMA, which uses a broadcast collective in a heterogeneous 1D data partition.

Also, here we discusses the automatic building and evaluation of the τ–Lop communication model of a data parallel kernel. As τ–Lop cost expressions on a heterogeneous platform rapidly become complex enough to be unmanageable, the τ–Lop library is used. Needless to say, the inputs of the library are:

- τ–Lop parameters for a specific platform.
- Kernel communications.
- Data space partition.

The output is the cost of the kernel communications in the platform in terms of time. Integration of the library and the Beaumont algorithm is straightforward. For each and every generated partition, the Beaumont algorithm invokes the library interface to compose and evaluate the associated cost expression Θ. A pair of kernels are used throughout the paper as conduit examples, namely the two dimensional wave equation and the parallel dense matrix multiplication, *Wave2D* and *SUMMA* respectively. They were chosen by their opposite communication needs. SUMMA has a communication complexity $O(N^2)$, while *Wave2D* shows just $O(N)$. Following, we discuss how the library builds and evaluates both.

Wave2D Model Building

The Wave2D kernel simulates a wave behavior under boundary conditions using the technique of finite differences. The 2D wave equation is 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 $t \in [1, T)$, with N the size in elements of the mesh and T the number of considered iterations. Along time, $u(x, y, t+1)$ is given by New, generated from its previous instances, the matrices Cur and Old, $u(x, y, t)$ and $u(x, y, t-1)$ respectively, according to the stencil in the right side of Figure 5.4, which illustrates a data partition between $P = 8$ processes. The point-to-point transmissions T from

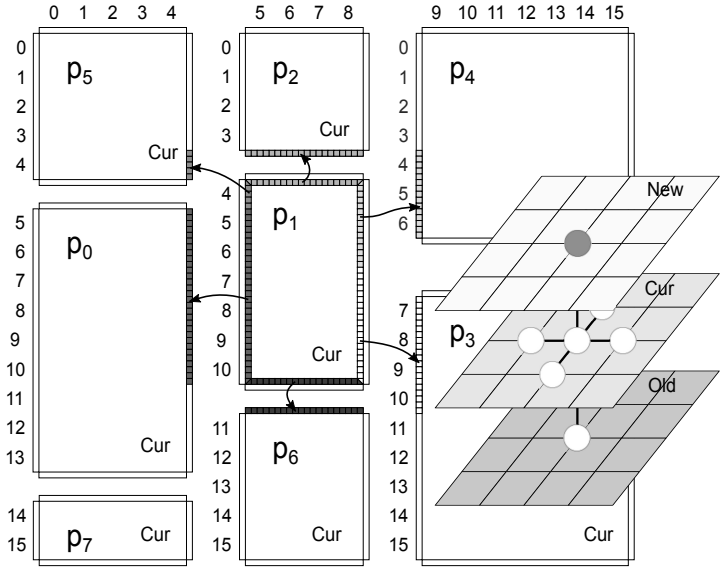


Figure 5.4: A 2D partition for solving the wave equation. Each process recomputes its rectangles New, Cur and Old along time. The computation stencil is $New(i,j) = 2(1-2C^2)Cur(i,j) - Old(i,j) + C^2Cur(i-1,j) + C^2Cur(i+1,j) + C^2Cur(i,j-1) + C^2Cur(i,j+1)$. Note that it imposes communications in Cur. Process p_1 sends its perimeter elements of Cur, which will form the halo of its neighbors.

p_1 to its neighbors are shown. For instance, if η_i denotes the neighborhood of p_i in the clockwise order, then $\eta_1 = \{2, 4, 3, 6, 0, 5\}$. Non-blocking sends (and receives) are used. Because of the computational balance, all the processes start their transmissions at once after the computing stage, so the cost per iteration allocated to a process p_i will be:

$$\Theta_p = \mathop{\Vert}_{j \in \eta_p} T^{c(j)}(m(j)) \tag{5.5}$$

where η_p is the set of neighbor processes of p, $m(j)$ the size of the message sent to neighbor p_j, and $c(j)$ the channel used to send the message. We can appreciate that the transmissions from the processes P are progressing concurrently in the channel $c(j)$. Is determined the τ–Lop cost of I iterations of the algorithm as:

$$\Theta^{W2D} = I \times \left[\mathop{\Vert}_{p=0}^{P-1} \Theta_p \right] \tag{5.6}$$

The theoretical model Θ^{W2D} from Algorithm 5 of the expression (5.6)

estimates its cost with $P = 8$ processes and $M = 4$ nodes. This model, to be evaluated must be expressed as a function of the machine specific parameters. Overhead o and Transfer Time L are the parameters of the model measured in the specific platform and used to evaluate the cost expression. Thus, the complexity grows with the number of processes in a partition, needing a automatic approach for this. *Processes* in the vector p are created with the appropriate rank, and rank-to-node mapping described in the vector $nodes$. The inner for loop traverses the set of neighbors processes η_p for every source process and builds the transmissions in a $TauLopSequence$ object, which can be used to for a concurrency object *conc* of type *TauLopConcurrent* to represent concurrency of sequences of transmissions. Function *getBoundarySize(src, dst)* returns the number of halo bytes to be interchanged by processes ranks *src* and *dst*. These sequences are composed of $Transmission$, which are composed by source and destination processes, the message size m, and the number of concurrent transmissions as τ. The communication channel c is figured out from the information of the source and destination processes. Hence, every transmission is modeled as $1 \parallel T^c(m) = [o^c(m) + L^c(m, 1)]$. Each per-process *TauLopSequence* objects are added to an unique concurrent *TauLopConcurrent* object according to the expression (5.6), which is finally evaluated to predict the overall communication cost.

We shown the accuracy of the Wave2D estimation in *Figure 5.5*. There are two lines, which are the times estimated by τ–Lop tool with the parameters, and the real times of the communication. The platform was described in 5.3.

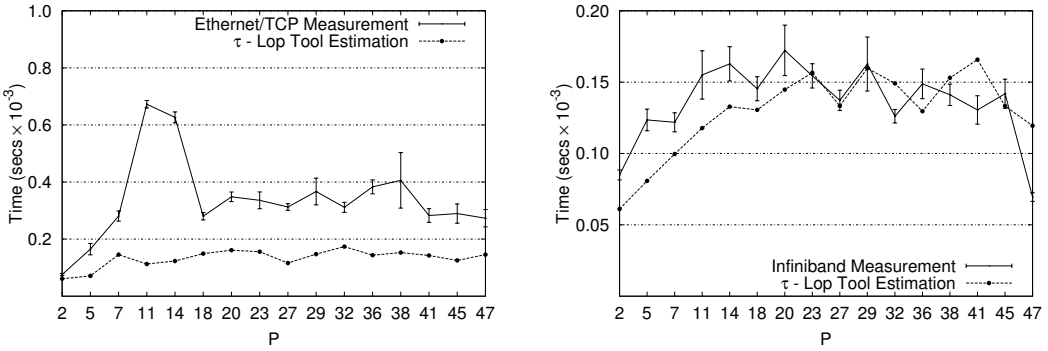
Algorithm 5 Wave2D kernel: Cost modeling and evaluation

```
int P = 8;
int nodes = {0, 1, 2, 3, 0, 1, 2, 3};
Process *p [P];

for rank ∈ {0, ..., P − 1} do
   [rank] = new Process (rank, nodes[rank]);
end for
TauLopConcurrent *conc = new TauLopConcurrent ();

for src ∈ {0, ..., P − 1} do
      int *η_p = getNeighbors(src);
      for dst ∈ {η_p} do
         m = getBoundarySize(src, dst);
        *seq = new TauLopSequence ();
         seq→add (new Transmission (p[src], p[dst], m, τ=1));
         conc→add (seq);
   end for
end for

TauLopCost *tc = new TauLopCost ();
conc→evaluate (tc);
double t = tc→getTime ();
```



(a) Wave2D TCP

(b) Wave2D Infiniband

Figure 5.5: Wave2D measured communication cost compared to the tool estimation on Infiniband (left) and Ethernet/TCP (right) networks. The plots represent the communication time per iteration of the kernel for a matrix size of N=512 elements

SUMMA Model Building

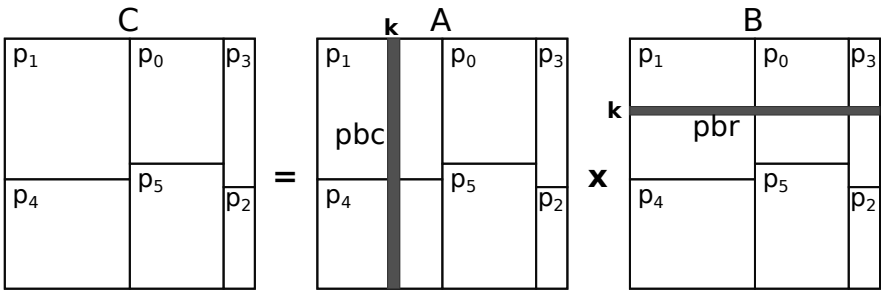


Figure 5.6: The SUMMA algorithm on a heterogeneous platform with $P = 6$ processes. A rectangle of different size is assigned to each process. White rectangles are assigned to processes running on the node 0, and grey rectangles to that on the node 1. The figure shows the iteration k. p_1 sends its part 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 SUMMA algorithm computes the dense matrix multiplication $C = A \times B$. For simplicity, we assume square matrices. The block is the unit of computation, and its size is set to take advantage of the hardware memory hierarchy. The elements of the matrices are grouped into blocks of size $b \times b$. With this we can determine the size of the matrices as $N \times N$ blocks, to be partitioned into P non-overlapping rectangles assigned to the P processes.

SUMMA executes in N iterations. Figure 5.6 shows an iteration k in an example of the algorithm with $P = 6$ processes.

In iteration k, every process computes partial results for the blocks on its assigned rectangle of C, as $c_{ij} + a_{ik} \times b_{kj}$. Needless to say, previously 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 $c_{ij} = \sum_{k=1}^{N} a_{ik} \times b_{kj}$. Each iteration k is composed of three stages:

1. Processes owning the k-th *pbc* of the matrix A send the blocks to the processes in the same row,

2. Processes owning the k-th *pbr* of the matrix B send the blocks to the

processes in the same column

3. Each process p_i updates the blocks in its assigned rectangle of matrix C.

There are different communication patterns as *point-to-point* for two processes communication, *ring* for communication between processes in the same row or column, and *broadcast* for communication between a set of processes in the same column that is possible because columns have the same width in the Beaumont algorithm. The communication of sending and receiving blocks of the *pbc* and *pbr* can be evaluated under these patterns. We will evaluate *point-to-point* and *ring* patterns for the kernel. Next, the analytical model of the SUMMA kernel Θ^{SUMMA} will be built. As an example, the point-to-point pattern of communication is modeled here. The Ring and broadcast patterns are similar. As first step, we represent the horizontal communication cost of sending *pbc* blocks in matrix A.

The cost of the sequence of transmission is represented as $\Theta^{pbc}_{k,p}$. This is performed by each process p holding the *pbc* to its overlapping processes (in set η_p). These transmissions goes through a specific channel c_j with a specific message size m_j.

$$\Theta^{pbc}_{k,p} = \sum_{j=1}^{card(\eta_p)} T^{c_j}(m_j) \tag{5.7}$$

P^{pbc}_k is the set of all the processes holding the *pbc* in the iteration k, which is composed in Fig. 5.6, for instance, by p_1 and p_4. Θ^{pbc}_k is the global cost of all of them.

$$\Theta^{pbc}_k = \parallel_{p \in P^{pbc}_k} \Theta^{pbc}_{k,p} \tag{5.8}$$

Next, is time to model vertical communication cost of sending the blocks of *pbr* in matrix B. Cost of transmissions in a column *col* is modeled as $\Theta^{pbr}_{k,col}$, which

represents a sequence of point-to-point transmissions from the process holding the *pbr* to the rest of processes in the same column, represented by the set η_{col}.

$$\Theta_{k,col}^{pbr} = \sum_{j=1}^{card(\eta_{col})} T^{c_j}(m_j) \tag{5.9}$$

P_k^{pbr} is the set of all the processes holding the *pbr* in the iteration k. This set is composed in Fig. 5.6, for instance, by p_0, p_1 and p_3. Θ_k^{pbr} represents the concurrency and the global cost of the sequences of transmissions on the columns of the partition in the iteration k.

$$\Theta_k^{pbr} = \parallel_{p \in P_k^{pbr}} \Theta_{k,col(p)}^{pbr} \tag{5.10}$$

Finally, we can establish the total cost of the SUMMA algorithm as:

$$\Theta^{SUMMA} = \sum_{k=0}^{N-1} \left[\Theta_k^{pbc} + \Theta_k^{pbr} \right] \tag{5.11}$$

Algorithm 6 SUMMA kernel: Cost modeling and evaluation

```
int P = 8;
int nodes = {0, 1, 2, 3, 0, 1, 2, 3};
double t = 0.0;

for k ∈ {0, N − 1} do
    // Horizontal communication (pbc on matrix A)
    TauLopConcurrent *conc_H = new TauLopConcurrent ();
    for src ∈ {0, ..., P − 1} do
        if hold_pbc(src, k) then
            TauLopSequence *seq = new TauLopSequence ();
            for dst ∈ {0, ..., P − 1} do
                m = overlapSize(src, dst) × b^2  if m > 0 then
                    seq→add (new Transmission (p[src], p[dst], m, τ=1));
                end
            end
            conc_H→add (seq)
        end
    end
    TauLopCost *tc_H = new TauLopCost ();
    conc_H→evaluate (tc_H);
    double t_pbc = tc_H→getTime ();

    // Vertical communication (pbr on matrix B)
    TauLopConcurrent *conc_V = new TauLopConcurrent ();
    for src ∈ {0, ..., P − 1} do
        if hold_pbr(src, k) then
            TauLopSequence *seq = new TauLopSequence ();
            for dst ∈ {η_col(src)} do
                if src ≠ dst then
                    m = getColumnWidth(src) × b^2;
                    seq→add (new Transmission (p[src], p[dst], m, τ=1));
                end
            end
            conc_V→add (seq)
        end
    end
    TauLopCost *tc_V = new TauLopCost ();
    conc_V→evaluate (tc_V);
    double t_pbr = tc_V→getTime ();

    // Accumulate horizontal and vertical times
    t += t_pbc + t_pbr
end
```

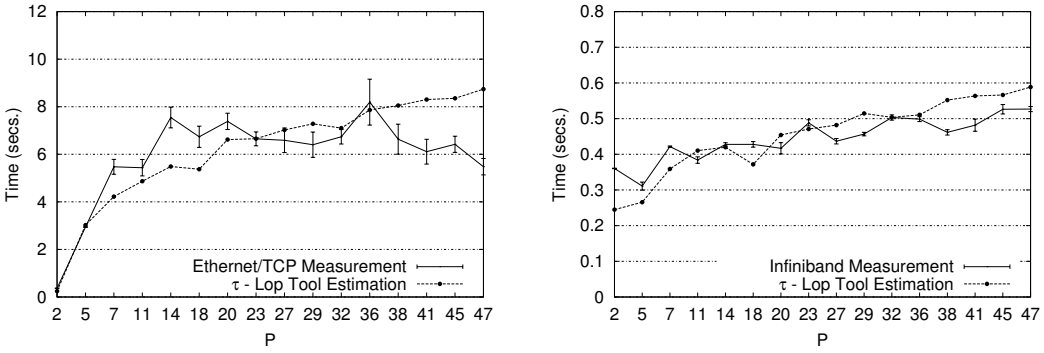
The kernel of SUMMA kernel by the τ–Lop model is modeled in Code 6, which evaluates the global communication cost of Θ^{SUMMA}.

Horizontal communication of the *pbc* blocks in each iteration k is achieved using point-to-point transmissions. As we know, we must build a $TauLopConcurrent$ object which contains *pbc* communications. This $TauLopConcurrent$ is created by the sequences of transmissions of the *pbc* blocks. Each $Source(src)$ process that holds the *pbc* creates a sequence *TauLopSequence* as *seq*. Every *Transmission* to an overlapped destination processes *dst* is added to its sequence *seq*. The size of *Transmissions* is obtained using the function $overlapSize(src, dst)$ and the size of a block b^2. At the end, we evaluate all the concurrent communications in $conc_H$ of each process holding the *pbc*. Thus, we obtain a time named as t_{pbc} in terms of time.

Vertical communication of the *pbr* in each iteration k is also point-to-point. In each column, a process performs the transmissions in concurrency with one process of the rest of columns. Every process holding *pbr* blocks in the iteration k (function $hold\_pbr(src, k)$) perform a point-to-point transmission to the rest of the processes in its column (set $\eta_{col}(src)$). Here, the size of the message of the transmission m is the width of the column, obtained by $getColumnWidth(src)$ function and the size of a block b^2. *Transmissions* are added to sequential objects, which will compose the concurrent object $conc_V$. The number of sequential objects are equal the number of columns. This is because transmissions in each column progress concurrently. As in Horizontal communication, the concurrent communications in $conc_V$ are evaluated and time t_{pbr} obtained in terms of time.

Total estimated communication time of the SUMMA algorithm is the sum of vertical and horizontal communication represented by t_{pbc} and t_{pbr} over the number of iterations N.

We shown the accuracy of the SUMMA estimation in *Figure 5.7*. There are two lines, which are the times estimated by τ–Lop tool with the parameters, and the real times of the communication. The platform was described in 5.3.



(a) SUMMA TCP

(b) SUMMA Infiniband

Figure 5.7: SUMMA measured communication cost compared to the tool estimation on Infiniband (left) and Ethernet/TCP (right) networks. The plots represent the communication time per iteration of SUMMA for matrix size of N=512 blocks of b=32 double precision elements

5.3 Parameters Measurement for τ–Lop Model

The work of this section has been developed in an accurate manner, and is the basis for the construction of the automatic tool. Therefore, we consider necessary the detailed explanation of this process.

As we have explain in previous Section 5.1, the model is composed by different parameters. $L^c(m, \tau)$, known as the *transfer time*, and $o^c(m)$, known as the *message overhead*. These parameters must be measured in an independent way for each platform. Also, they depends on the network channel c. The parameters are measured using an amount of repetitions, with a maximum of 1000 repetitions. Each repetition makes a transmission of a message size m, with its own time. With this transmission time we can establish which amount of time is related to a specific parameter. It is not necessary to execute all the repetitions. This is because we also implement a trust factor of $c = 0.95$, so when the time difference between an amount of repetitions, normally 30, is bigger than this factor, it automatically stops. Both parameters are measured for a message size value m. Also, *transfer time* is measured for a specific τ value.

In the case of the *message overhead*, parameters are measured using a PingPong/RTT data exchange of a $m = 0$ message size, so transfer time is $L_j^c(0,1) = 0$. This communication protocol depends on the channel that we are using in the measurement. These different types of communication are shown in Figure 5.8, which are Eager and Rendezvous. Thus, in shared memory $o^0(m)$ the protocol used is Eager for the whole range of m message sizes, which is implemented using the primitives *MPI Send()* and *MPI Recv()* for all ranges of messages. This communication is composed of two point-to-point transmissions. The cost of each transmission is the addition of the overhead and the sequence of s transfer to reach destination. To measure this we create a job with $P = 2$ processes in one node, distributed each of them on different sockets. We can define the cost of the operation as:

$$\mathrm{o}^0(m) = 2 \times \frac{RTT^c(0)}{2} - \sum_{j=0}^{s-1} L_j^c(0,1) = \frac{RTT^c(0)}{2} \tag{5.12a}$$

In case that we are using a network channel as TCP or Infiniband we also estimate $o^1(m)$ under the Eager protocol, but when the message size reaches a threshold size H, the protocol change to Rendezvous. This protocol is implemented using *MPI Ssend* primitive and before the data transmission starts the sender process sends a RTS (Request to Send) to the receiver, which responds with a CTS (Clear to Send) when is ready, avoiding the sender to flood the receiver. This way, only one transfer is needed, leading to a cost of:

$$\mathrm{o}^1(m) = Ping^c(0) - \sum_{j=0}^{s-1} L_j^c(0,1) = Ping^c(0) \tag{5.13a}$$

RTT^c operation is not used for the rendezvous protocol because it would add a second point-to-point response message by the process P_j. Thus, it can start such response message RTS before the process P_i finished the reception of CTS. This overlapping would lead to a wrong overhead estimation.

For measure the *transfer time*, we need to create specific groups of processes, synchronize them and then start the measurement. These number of processes

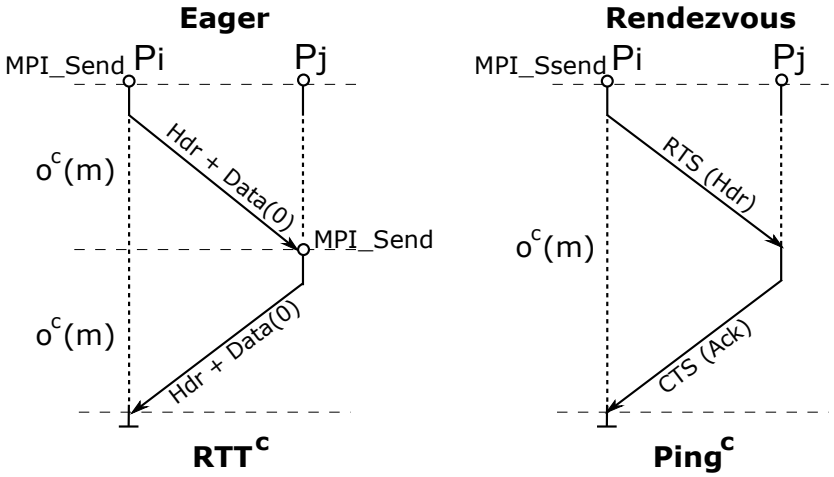


Figure 5.8: RTT^c and $Ping^c$ operations to measure overhead $o^c(m)$ parameter under eager and rendezvous communication protocols respectively. These operations are used in shared memory and any network communication channels.

establish the τ value, that is, the number of concurrent transmission progressing through the channel. They way we did this is shown in Algorithm 7.

Algorithm 7 Groups creation to measure transfer time parameter

```
int me;
MPI Group group;
MPI Comm group (comm, &group);
MPI Group newgroup;
int *ranksv = (int *) malloc (sizeof(int) * p);

for int P = 2; i<numprocesses; P++ do
    for int i = 0; i<P; i++ do
        ranksv[i] = i;

    end
    MPI Group incl(group, p, ranksv, &newgroup);
    MPI Group rank(newgroup, &me);
    MPI Comm create(MPI COMM WORLD, newgroup, &newcomm);

    if me != MPI UNDEFINED then
        MPI Barrier(newcomm);

        Here we measure the transfer time parameter
        measureTransferTime(data);
        MPI Comm free(&newcomm);

    end
    MPI Barrier(MPI COMM WORLD);
    MPI Group free(&newgroup);
    free(ranksv);

end
MPI Barrier(MPI COMM WORLD);
MPI Group free(&group);
```

In MPICH and Open MPI, communication between processes in shared memory progresses through intermediate buffers, so two transfers are needed to reach the destination buffer. Thus, a Ring operation in defined for these exchanges. This operation is implemented using *MPI Send* and *MPI Recv* when τ=1 and *MPI SendRecv* when $\tau > 1$. This entails a transmission to process P_{i+1}, and a transmission from process P_{i-1}, with wraparound and a wait operation until both complete. These different forms to measure the cost is because with $P = 2$ and $\tau = 1$ using a *MPI SendRecv* transmission could lead to a buffer re-usage in future repetitions, so the time is influenced by this. To measure this,

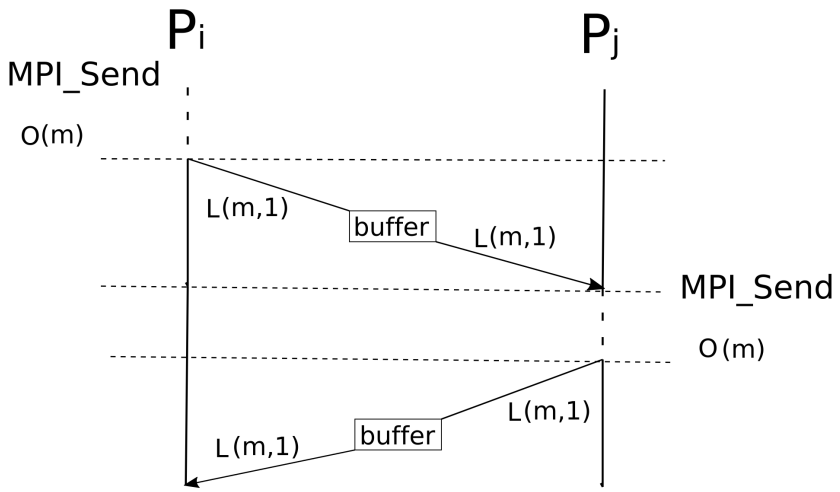


Figure 5.9: RTT^c operation used in the measure of Transfer Time parameter under shared memory transmissions. This operation is used when $\tau = 1$.

we create a job on the platform with P processes in one node, distributed with Round Round method along the sockets. We can define the cost of transfer time in shared memory when τ=1 as:

$$\mathrm{R}TT^c(0) = 2 \times [o^0(m) + 2L^0(m,\tau)] \tag{5.14a}$$

$$\mathrm{L}^0(m,\tau) = \frac{\frac{RTT^c(0)}{2} - o^0(m)}{2} \tag{5.15a}$$

Also, the cost when τ>1 is calculated as:

$$\mathrm{R}ing^0_\tau(m) = o^0(m) + 2L^0(m,\tau) \tag{5.16a}$$

$$\mathrm{L}^0(m,\tau) = \frac{Ring^0_\tau(m) - o^0(m)}{2} \tag{5.17a}$$

Algorithm 8 Parameter measurement of transfer time under shared memory

```
int maxreps = 10000;
int stop = 0;
int reps = 0;
double t = 0.0;
int rank, source, dest;

while (stop == 0) and (reps < maxreps) do
    dest = (rank + 1) % numprocs;
    source = (rank - 1) % numprocs;
    if source < 0 then
        source = numprocs - 1;

    end
    if numprocs=2 then
        if rank = 0 then
            T[reps] = MPI Wtime();
            MPI Send(buffer, M, MPI BYTE, dest, 0, comm);
            MPI Recv(buffer, M, MPI BYTE, dest, 0, comm, MPI Status Ignore);
            sum += T[reps] = MPI Wtime() - T[reps];

        end
        else
            MPI Recv(buffer, M, MPI BYTE, dest, 0, comm, MPI Status Ignore);
            MPI Send(buffer, M, MPI BYTE, dest, 0, comm);

        end
    end
    else
        T[reps] = MPI Wtime();
        MPI Sendrecv(buffer, M, MPI BYTE, dest, 0, buffer, M, MPI BYTE,
        source, 0, comm, MPI Status Ignore);
        sum += T[reps] = MPI Wtime() - T[reps];

    end
    t = sum / reps;

    // Here we calculate the transfer time parameter as define in the formulation
    TransferTimeAdapt(t, M, overhead);

end
```

As shown in Algorithm 8, we can appreciate how the transfer time is measured under the different forms. Thus, the algorithm behaviour is shown in *Figure 5.9.*

The communication between processes through TCP and Infiniband networks requires three intermediate transfers to reach the destination buffer. The first and last transfer will progress through shared memory, with a cost measured in 5.15a and 5.17a. To measure the transfer time parameter under network channels we use *MPI Send* and *MPI Recv*. 2τ processes are mapped using Round Robin in two nodes, so contiguous processes will run in different nodes. This communication operation has two stages. The first step is when each process P_i sends to the process P_{i+1}. The second step is when each process P_i receive from the process P_{i-1}. The overlap of copy to the NIC internal buffer and transmission through the network is unavoidable, so we do not considered this because of its random behaviour. The cost of a TCP transfer can be determined as:

$$\mathrm{R}ing^1_\tau(m) = 2 \times [o^1(m) + 2L^0(m,\tau) + L^1(m,\tau)] \tag{5.18a}$$

$$\mathrm{L}^1(m,\tau) = \frac{Ring^1_\tau(m)}{2} - o^1(m) - 2L^0(m,\tau) \tag{5.19a}$$

In Infiniband is quite different because it use RDMA mechanism which involve that shared memory transfer time is not taken into account. The cost of a Infiniband transfer can be determined as:

$$\mathrm{R}ing^1_\tau(m) = 2 \times [o^1(m) + L^1(m,\tau)] \tag{5.20a}$$

$$\mathrm{L}^1(m,\tau) = \frac{Ring^1_\tau(m)}{2} - o^1(m) \tag{5.21a}$$

The platform where we measured the parameters is a cluster composed of NVidia Tesla M2075 GPUs and Intel Xeon E5649 multi-core (8-cores and 12-cores) CPUs. Range of processes is from $P = 2$ running in one node to P = 47 in sixteen nodes. Processes are deployed in the available computing resources. Two types of network are used, a 40 Gbps QDR Infiniband and a 1Gbps Ethernet/TCP . The communication library is Open MPI. We shown the accuracy of the communication cost predictions of the τ–Lop Tool in composing

and evaluating the Wave2D and SUMMA kernel communications in previous Section 5.2.3.

6

IMPLEMENTATION

We presented the problem of finding a load-balanced partition of a 2D data-space between a set of processes characterized by different computing power. Beaumont et al., 2001 proposed it as a formal optimization problem. They also proposed a heuristic base on *dynamic programming* which builds a column-based partition. This heuristic minimizes the communication volume using the sum of the half perimeters as a metric of the resultant partition rectangles which are assigned to each computation unit.

The main feature in this metric is that the communication volume is proportional to the perimeter of the resultant rectangles, which is minimized when rectangles become as square as possible. We will study this metric discussing shortcoming of the solutions. Later, we present a new metric based on the τ–Lop communication performance model. We focus on the problem of non-uniformly partitioning a 2D data space between processes with different capabilities on a heterogeneous system, with the goal of minimizing the communication cost.

6.1 Beaumont Optimization Solution

Beaumont et al., 2001 formulate the partitioning of a kernel data space in an optimization problem as a integer program. The program inputs are the number of processes P, the size of the 2D data space (N^2) and the speed array $S = \{s_1, s_2, \ldots, s_P\}$, where s_i is the relative speed of the process p_i. With this, the objective is to partitioning the data space with P non-overlapping rectangles with its own sizes $n_i = w_i \times h_i, 1 \leq i \leq P$, being proportional to the speed of the process. The problem is relaxed and reformulated in terms of a linear program with the following constraints:

$$\sum_{i=1}^{P} n_i = 1. \tag{6.1a}$$

Rectangles are non-overlapping and tile the unit square. (6.1b)

$$\frac{n_1}{s_1} = \frac{n_2}{s_2} = \cdots = \frac{n_P}{s_P}. \tag{6.1c}$$

Restrictions (6.1a) and (6.1b) stand for tiling the whole data space (unit square) with the P non-overlapping rectangles. Last restriction (6.1c) implies the non-uniform load balance of the partition, in which every process is assigned with a rectangle of an area in proportion to its speed. As a consequence, the execution time $t_i = n_i/s_i$ will be equal for every process. The linear programming optimization problem has now a solution that is later approximated by scaling up the unit square partition to the size of the problem $N \times N$. Thus, the goal is to find a partition of the data space which minimizes the communication cost.

Beaumont et al., 2001 proposed to minimize the next objetive function, which shows the sum of the *half perimeters* of the rectangles assigned to each process. It represents the total volume of communications, and is minimized as the rectangles become as square as possible.

$$\beta = \sum_{i=1}^{P} (w_i + h_i), \tag{6.2}$$

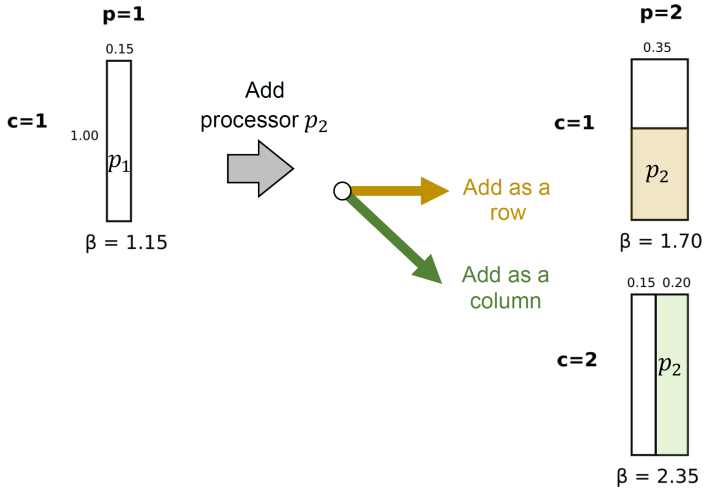
As Beaumont et al., 2001 demonstrated, this problem is NP-Complete. Thus, to solve this a geometrical heuristic to obtain a near optimal solution and an algorithm to find it is proposed. It consists of the following steps:

1. Order the relative speed vector S is such a way that: $s_1 \leq s_2 \leq \ldots \leq s_P$.

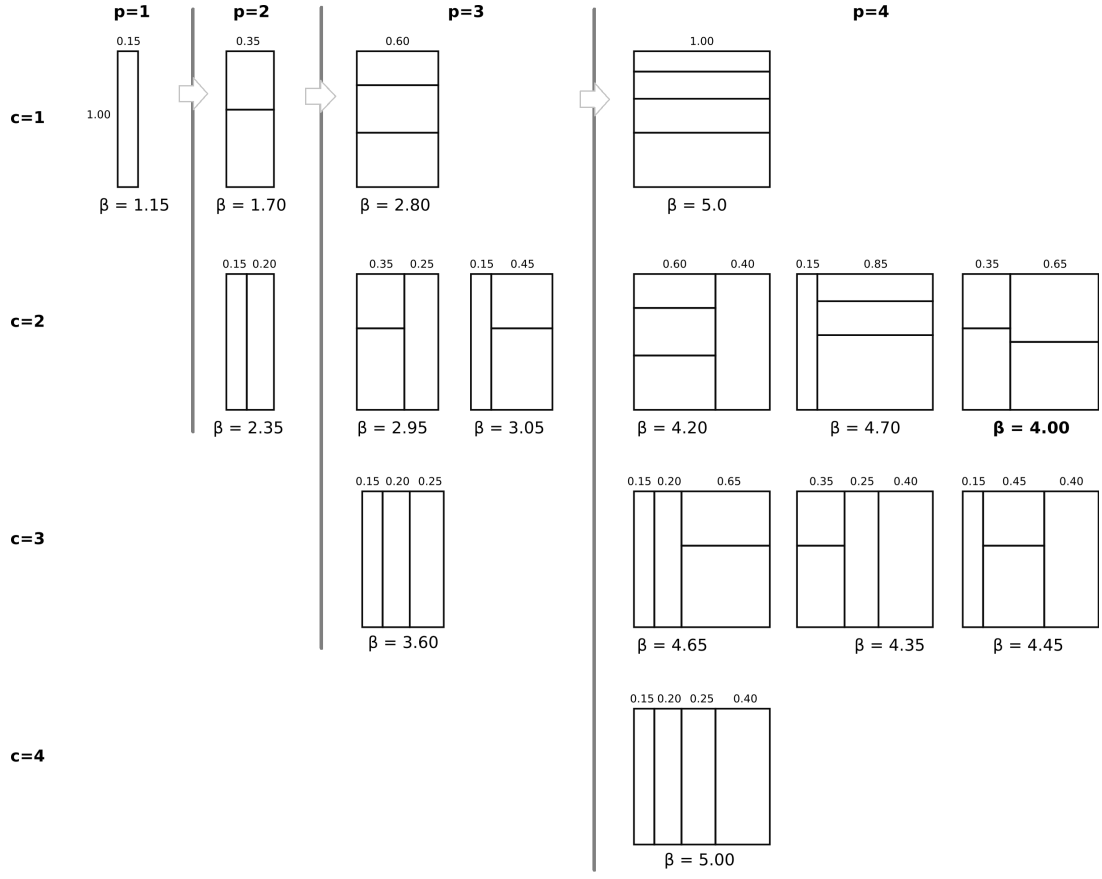
2. Using a *dynamic programming* technique, the algorithm generates a new set of arrangements by adding rectangles in the previous specified order in S. For each previously generated arrangement, the new rectangle of p_i is added to the last column (as a new row) by increasing its width, and also as a new column of height $h_i = 1$ and width $w_i = s_i$. The half perimeter metric is used to evaluate every additional arrangement.

3. A rollback is the last stage to decide the optimal composed arrangement based on the half perimeter metric.

The complexity of the algorithm is polynomial $O(p^3)$ and it provides the optimal shape of the rectangles for every process, the optimal number of columns and the optimal number of processes per column, using the half perimeter metric.



(a) First and second tiles



(b) Full tile space

Figure 6.1: The Beaumont partitioning algorithm at work. Here, it incrementally builds a partition space for $P = 4$ processes characterized by the ordered relative speed vector $S = \{0.15, 0.2, 0.25, 0.4\}$. Each step adds a process to each tiling of the previous partition space in two different ways.The result is that the number of candidate tilings doubles in each step. The communication volume metric decides the winner tiling, that of volume $\beta = 4.0$.

A simple example with $P = 4$ processes with relative speeds $S = \{0.15, 0.2, 0.25, 0.4\}$ distributed in $M = 2$ nodes of a hypothetical heterogeneous platform. This example is shown in Figure 6.1. Background colors represent the node, where every color is a different node. Each step in the algorithm (c, p) contains a set of arrangements built from every generated arrangements in the previous $(c-1, p-1)$ and $(c, p-1)$ steps, with p the number of processes already added to the partition and c the number of columns. The half perimeter metric function β is used to assign a value to every new arrangement. This value in the example is minimum in one of the partitions of the step $(c = 2, p = 4)$.

Beaumont algorithm deals with the problem of partitioning on heterogeneous platforms, minimization of the communication volume by using the half perimeter metric is oblivious to the following three points:

- Kernels use specific communication patterns, which are not captured by the communication volume metric. For instance, a finite-element PDE solver usually communicates every process halo to its nearest neighbors (boundary rectangles), while a matrix multiplication kernels communicates with all the processes in the same row and column. Intuitively hence each different kernel will have its different optimum partition. However, the half perimeter metric outputs the same partition for widely different kernels: SUMMA and Wave2D, for instance, share the same solution.

- Heterogeneous platforms are characterized for using different types of networks. Processes communicate using available these communication channels, as shared memory, high performance networks, and high throughput networks as those connecting clusters in a grid.

- Half perimeter metric represents the volume of point-to-point communication between boundary rectangles in the partition. Anyways, kernels implementations use advanced communication modes for improving its performance, as collective communication, which are not captured by the communication volume metric.

6.2 Using the τ–Lop Metric

The total volume of communication metric β disregards the real time cost of communications. It is necessary to implement a metric which captures the characteristics of the heterogeneous platform and the specific kernel communication patterns.

6.2.1 Modifying the Original Metric

We propose a *communication performance model* metric for being able to be output a more accurate communication cost estimation. A communication performance model provides with an analytic framework that represents communications as a parameterized formal expression described in 5.1. The evaluation of this expression determines the cost of the communication in terms of real time, as a function of system parameters.

The new objective is to replace the half-perimeter expression (6.2) with the new optimization based in finding the partition which minimizes its τ–Lop cost expression Θ. In terms of Figure 6.1, the full partition space generated by the Beaumont algorithm is now reevaluated tagging each tiling with its Θ value, superseding the total volume of communications metric β previously used. Thus, the goal is to automatically built and evaluate Θ.

τ–Lop metric is a function of three variables and hence could be notated as $\Theta(k, \pi, D)$, with k standing for the kernel, π for the target platform and D for the data partition. For readability, however, we will adopt a lighter notation when convenient. Instead, the half-perimeter metric is a function of just the data partition and hence it could also be represented as $\beta(D)$.

A simple study case which provides some insight on the application of the metrics $\beta(D)$ and $\Theta(k, \pi, D)$ is presented. A more complex but still easy to handle ordered speed vector $S = \{0.05, 0.05, 0.08, 0.1, 0.1, 0.12, 0.2, 0.3\}$ of with

Table 6.1: Cost results under the half-perimeter metric $\beta(D)$, where D belongs to the set of tilings generated by the algorithm of Beaumont applied to the speed vector $S = \{0.05, 0.05, 0.08, 0.1, 0.1, 0.12, 0.2, 0.3\}$ of with $P = 8$ processes. Adapted from Beaumont et al., 2001, only the lowest β value for the set of tilings in each step (c, p) is shown. Note that the optimum partition according to the metric contains $c = 3$ columns.

	p = 1	p = 2	p = 3	p = 4	p = 5	p = 6	p = 7	p = 8
c = 1	1.05	1.20	1.54	2.12	2.90	4.00	5.90	9.00
c = 2		2.10	2.28	2.56	2.94	3.50	4.38	5.76
c = 3			3.18	3.38	3.66	4.00	4.58	**5.50**
c = 4				4.28	4.48	4.78	5.20	5.88
c = 5					5.38	5.60	5.98	6.50
c = 6						6.50	6.80	7.28
c = 7							7.70	8.10
c = 8								9.00

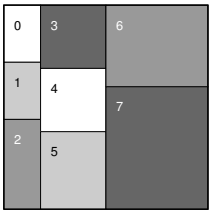
$P = 8$ processes will be used. The mapping of the processes on the nodes (which determines the communication channels) is given by $V = \{0, 1, 2, 3, 0, 1, 2, 3\}$, that is, process with rank p runs on node $V[p]$. We suppose a hypothetical platform of $M = 4$ nodes numbered from 0 to 3, with two available communication networks *TCP* and *Infiniband*. The latency L and overhead o which composes τ–Lop parameters characterizing the platform have been measured in Section 5.3. Regarding the kernel k, we evaluate SUMMA with two different communication patterns, namely point-to-point and Ring. The Wave2D kernel, however, allows only point-to-point communication.

Table 6.1 shows that the winner tiling under the half-perimeter metric β is an arrangement of $c = 3$ columns. Table 6.2 shows the communication times obtained by the Θ metric from the Wave2D and SUMMA kernels under different configurations of the example platform. Note that also three-columns winners are the rule here, but exceptions appear sometimes. The τ–Lop Θ metric captures the specific configuration of the platform and finds the actual best partition. As well, the cost returned by τ–Lop is a meaningful time cost estimation of the communications in the kernel on the platform, which allows us further optimization. Figure 5.7(a) shows the winner tiling of Table 6.1,

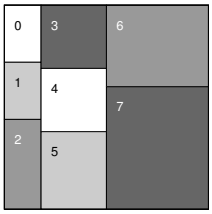
Table 6.2: Cost results under the τ–Lop metric for SUMMA and Wave2D kernels using two communication patterns on *TCP* and *Infiniband* networks. The metric is only applied to the partitions generated in the last step of $p = 8$ processes. Times are in μ-seconds.

	SUMMA IB P2P	SUMMA IB Ring	SUMMA TCP P2P	SUMMA TCP Ring	W2D IB P2P	W2D TCP P2P
c = 1	173244.8	180487.5	2330548.5	2697341.8	**32.12**	**366.33**
c = 2	64170.7	64168.6	836118.8	**801639.1**	36.29	398.19
c = 3	**54819.7**	**51752.4**	**826615.7**	805764.9	41.01	408.01
c = 4	65392.8	69677.8	893995.8	1071080.8	44.35	429.56
c = 5	87969.1	96925.4	1153902.4	1424385.5	61.52	395.08
c = 6	112666.2	126582.4	1492147.5	1825473.0	39.32	497.45
c = 7	141312.4	157749.0	1886612.5	2259679.5	36.05	649.06
c = 8	173244.8	180487.5	2330548.5	2697341.8	**32.12**	**366.33**

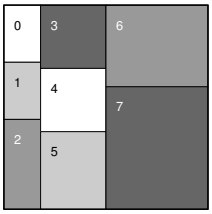
while Figure 5.7(b) to Figure 5.7(h) show the winner tiling of Table 6.2. The most relevant conclusion is that partitions differ for some combinations of kernel, specially for different networks and patterns, leading to wide different arrangements, while half perimeter metric is oblivious of them. Interestingly, Wave2D estimations for both Infiniband and TCP networks entail a 1D partition of the data space as the optimal for this example.



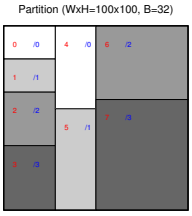
(a) Half perimeter partition



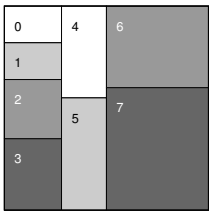
(b) SUMMA IB p2p



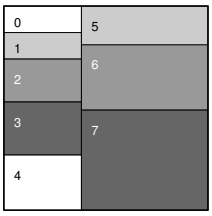
(c) SUMMA IB Ring



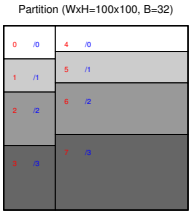
(d) SUMMA IB Bcast



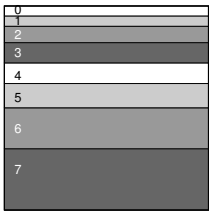
(e) SUMMA TCP p2p



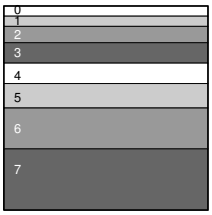
(f) SUMMA TCP Ring



(g) SUMMA TCP Bcast



(h) Wave2D IB p2p



(i) Wave2D TCP p2p

Figure 6.2: Optimum partitions according to Beaumont et al., 2001 original algorithm metric and τ–Lop metric results. In the case of τ–Lop metric, partitions are shown for every combination of kernel, network type and communication mode. Numbers indicate the ranks of the processes while background color represents the node of the process.

To have a clearer graphic visualization, we can apply the Θ metric to the partitioning process followed in the Figure 6.1. Therefore, in the following Figure 6.3, we can observe the different stages of the partitioning algorithm for both metrics (β, Θ). The winners of both metrics are highlighted. If we appreciate the different values for the winning partitions, we can see that for β metric the cost of the communications costs according to the Θ metric is higher than the winner of Θ metric.

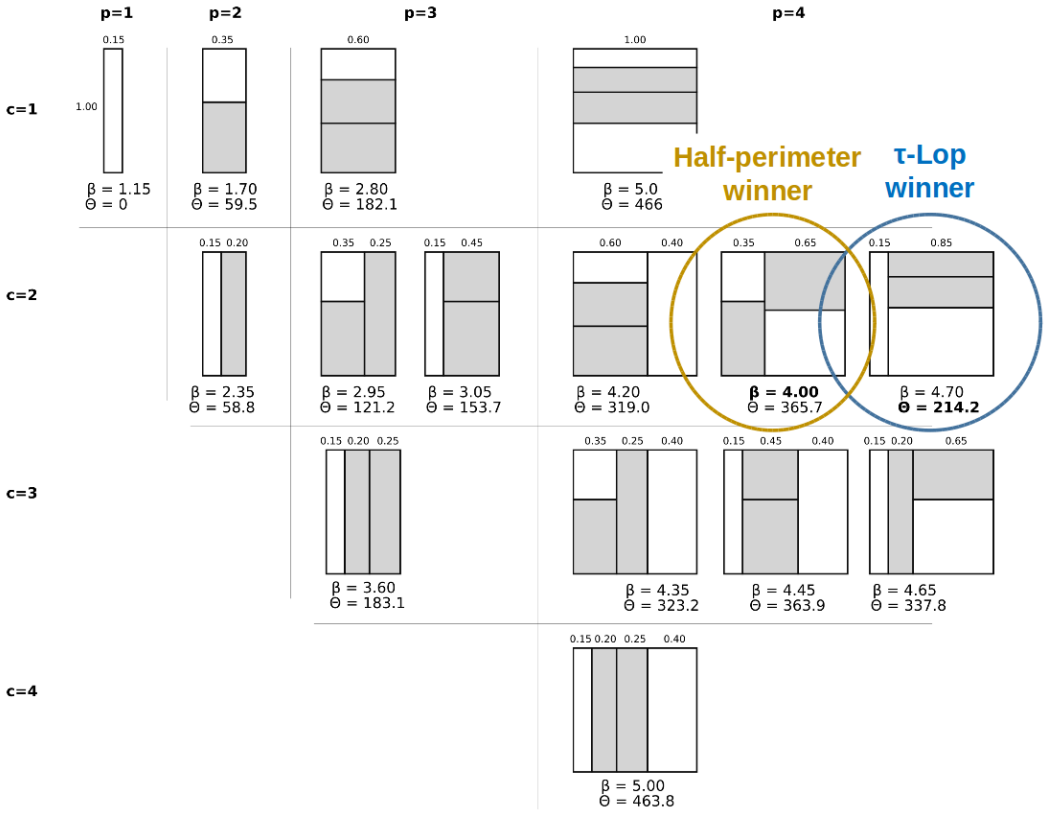
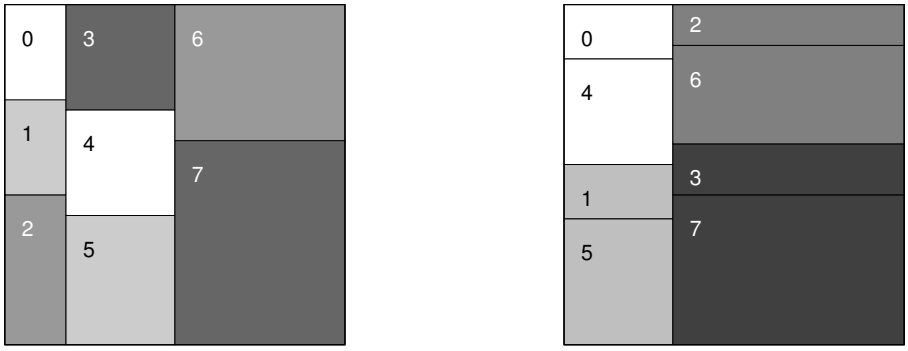


Figure 6.3: The Beaumont partitioning algorithm at work for different β and Θ metrics. It incrementally builds a partition space for $P = 4$ processes characterized by the ordered relative speed vector $S = \{0.15, 0.2, 0.25, 0.4\}$. Each step adds a process to each tiling of the previous partition space in two different ways.

6.2.2 Hierarchical ordering

There is a problem with the heuristic implementation using the half perimeter metric which does not reduce the sum of half perimeters values. It does not promote rectangles as square as possible, as can be deduced from the Figure 6.4, and hence it wrongly does not output optimum values. It just generates partitions with tiles which just grow in size to the right and down. This building path ignores the topology, that is, misses the tilings where the processes in the same machine are as close as possible. For this, we propose an alternative hierarchical order. Thus, we replaced the original growing order in the Beaumont algorithm by the hierarchical order.



(a) Original SUMMA TCP P2P Ring (b) Heuristic-based SUMMA TCP Ring

Figure 6.4: Output partition of the Beaumont algorithm using the half-perimeter metric compared to that of using the hierarchical order and the τ–Lop metric. The kernel is SUMMA using Ring pattern on a TCP network. Matrix size is $N = 512$.

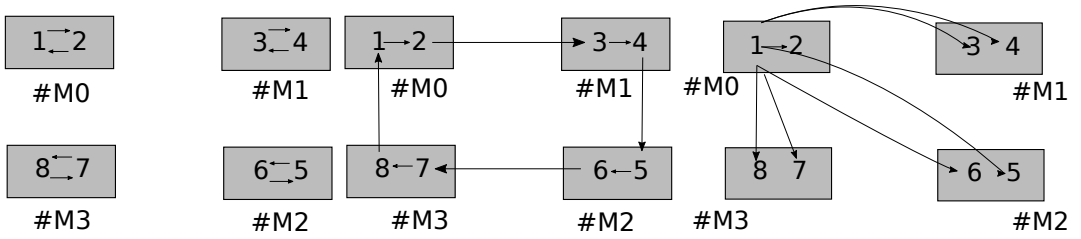
The hierarchical order works as follows. Regarding the example at the end of the section 5.1, the input vector $S = \{0.05, 0.05, 0.08, 0.1, 0.1, 0.12, 0.2, 0.3\}$ is now transformed in $S' = \{0.15, 0.15, 0.28, 0.4\}$, with $M = 4$ components, that is, the number of nodes. Every entry s'_i is the sum of the speeds of all processes in the node M_i, as determined by $V = \{0, 1, 2, 3, 0, 1, 2, 3\}$. After applying the Beaumont algorithm to S', resultant rectangles are horizontally split in proportional slides to processes speeds in each node.

7

RESULTS

We used *Fermi* as platform for ours tests, which is a highly heterogeneous platform with a set of between $M = 2$ and $M = 8$ twelve-core nodes, each with two connected GPUs. Nodes are 2-socket Intel Xeon E5649 processors (2.53 GHz) with two attached NVidia Tesla M2075 GPUs. The nodes are linked by both Infiniband QDR (40 Gbps) and TCP/Ethernet networks (1 Gbps). Processes in the same node communicate using shared memory, and processes running in different nodes communicate using the specific network in each experiment. We used two types of network because they represent different HPC platforms: while high performance networks as Infiniband are used in supercomputers, high throughput TCP networks usually connect clusters in grids.

Operating system is CentOS 6.5. 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 stencil Figure 5.4 of the Wave2D kernel. In the GPUs, cuBlas library for SUMMA and a self-



(a) P2P pattern (b) Ring pattern (c) Broadcast pattern

Figure 7.1: Example of communication patterns used in the tests made for both metrics (Θ and β). The example represent $M = 4$ nodes and $P = 8$ processes in a sequential mapping. In case of *Broadcast*, only one step is shown, where process $p = 1$ sends its data to all the processes

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*. Non-blocking point-to-point communication primitives are used in Wave2D.

We tested both metrics (Θ and β) under three different communication patterns, which are *Ring, P2P and Broadcast*. The behaviour of these patterns is shown in Figure 7.1 as an example. We have $P = 8$ processes distributed across $M = 4$ nodes in a sequential mapping. *Broadcast* pattern performs a communication in which each process sends data to the rest of processes, and also receives data from all of them. *P2P* pattern communicates two processes to each other, sending and receiving data between them. Last, *Ring* pattern performs the communication in a way that a process receives data from its previous and sends to its next.

Similar to Figure 1.1, we consider platform configurations with between five and twenty processes of different computing power. Table 7.1 shows the different configurations of CPU and GPU processes which have been actually tested. CPU processes execute in different number of cores using OpenMP threads. The M column represents the number of nodes used. Columns $c = \kappa$ indicate the number of processes in the row with κ cores assigned. The point here is that having

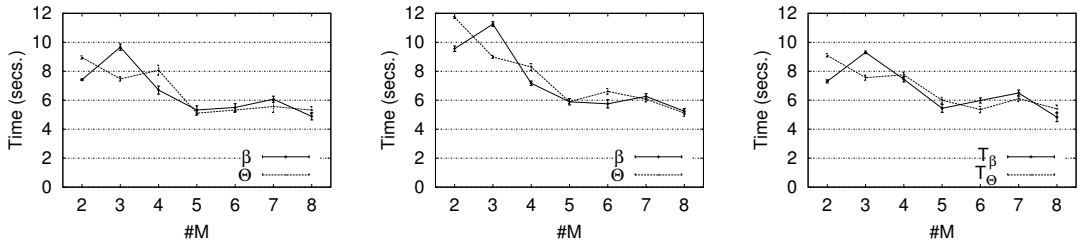
Table 7.1: Characterization of the set of nodes layouts used in the evaluation. They are provided by the SLURM scheduler.

Name	M	c=11	c=10	GPU	P
M2	2	1	1	3	5
M3	3	2	1	4	7
M4	4	2	2	6	10
M5	5	3	2	7	12
M6	6	3	3	9	15
M7	7	4	3	10	17
M8	8	4	4	12	20

processes with different computational capabilities increases the heterogeneity of the configuration. Column *GPU* indicates the number of processes running on GPUs. *P* column is the number of total processes, and it is the summation of the former three columns. In this platform 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 computational model of a GPU process as discussed by Zhong et al., 2015.

We ensure that the results are reliable and reproducible that have been proven by experiments. We indicate the spread of data by showing the standard deviation around the mean, that in some cases is imperceptible in the plots (specially for the Wave2D case, which is executed for a large number of steps). A kernel is executed by a set of processes, each proceeding as a sequence of communication and computation stages. For every individual measurement of a communication stage, the maximum time from all the processes involved in the communication is taken, under the assumption of that this maximum time is the actual total time of the communication. The computation is balanced using computational performance models as we explained in Section 1. Hence, we assume that all processes start the communication stage at the same time. The showed times are the mean of a set of measurements for any combination.

Figure 7.2 shows communication performance of SUMMA kernel on two distinct partitions. These partitions are generated by the Beaumont algorithm,



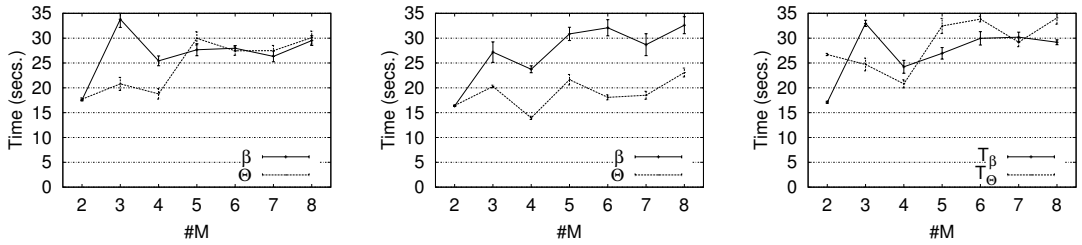
(a) SUMMA IB P2P 512 (b) SUMMA IB Ring 512 (c) SUMMA IB Bcast 512

Figure 7.2: Comparison of SUMMA measured communication times on winners partitions under the metrics β and Θ respectively. Measurements are taken along the complete execution of the kernel for increasing number of nodes (and processes) on an Infiniband network and different communication modes (point-to-point, Ring and Broadcast). Matrices size is $N = 512$.

by two different metrics, which are the winners under the half-perimeter (β) and the τ–Lop (Θ) metrics. There are not differences between results in Infiniband network. This is partially due to the S predefined order, which guides the generation of tilings as the algorithm progresses. At this point, it is needed to clarify that the times obtained for the range of nodes M in the x-axis are not smooth and linear, because in every case the partitions generated are completely different given the configuration of processes running on computational resources described in table 7.1.

We observed a special behavior under τ–Lop metric winners. There is a wide difference in performance between communication channels, as TCP and shared memory, τ–Lop winners show that closer rectangles are assigned to processes running on the same node. This grouping benefits the performance of the Ring communication pattern, because such arrangements reduce the network communication.

As we said in the previous Section 6.2.2, there is a problem with the heuristic implementation using the half perimeter metric which does not reduce the sum of half perimeters values. It does not promote rectangles as square as possible so we proposed an alternative hierarchical order with the performance effects shown in Figure 7.3. This hierarchical order allows to decrease the complexity of the



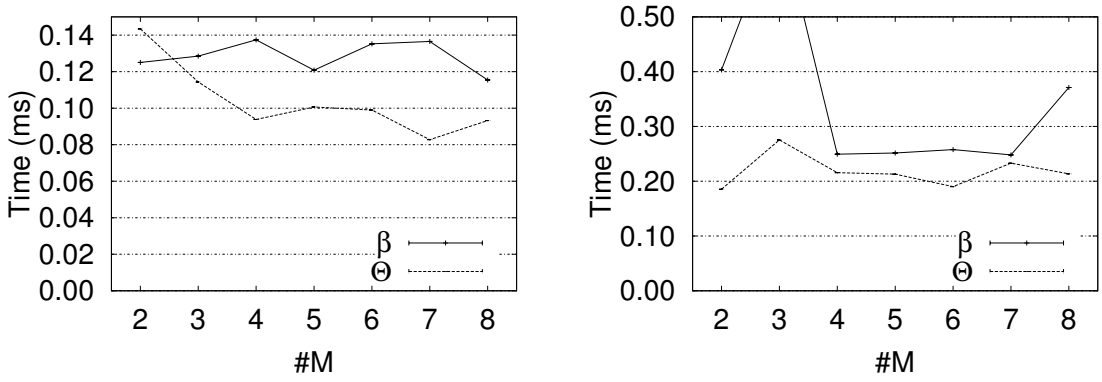
(a) SUMMA TCP P2P 512 (b) SUMMA TCP Ring 512 (c) SUMMA TCP Bcast 512

Figure 7.3: Comparison of SUMMA measured communication times on winners partitions under the metrics β and Θ respectively. This time, the tiling space has been generated using the grouping processes heuristic. Measurements are for increasing number of nodes (and processes) on a TCP network and different communication modes (point-to-point, Ring and Broadcast). Matrices size is $N = 512$.

Beaumont algorithm from $O(P^3)$ to $O(M^3)$, where usually $M << P$.

Additionally, we tested that in case of SUMMA on Infiniband networks, the RDMA mechanism discourages the usage of the hierarchical order, because it promotes the horizontal communication between processes in every two nodes. That is, in Figure 6.4 we can see that processes p_0 and p_4 running on node M_0 communicate concurrently the *pbc* to the processes p_2 and p_6 running on node M_2. The consequence is that sender processes flood the destination node, degrading the performance because of contention, with no overall appreciable benefit over the β metric. Note, as well, that the discussion about behaviors of TCP and Infiniband networks communications is partially derived from the τ–Lop cost modeling and estimations, before actual performance measurements, which in our view is an important contribution of this work.

We can observe the different output partitions of the Beaumont algorithm using half-perimeter metric and τ–Lop metric under the hierarchical order for the previous example in Figure 6.4. Serving SUMMA on Infiniband networks, the RDMA mechanism discourages the usage of the hierarchical order. This is cause it promotes the horizontal communication between processes in every two nodes. Figure 6.3*(b)* shows processes p_0 and p_4 running on node M_0 communicate



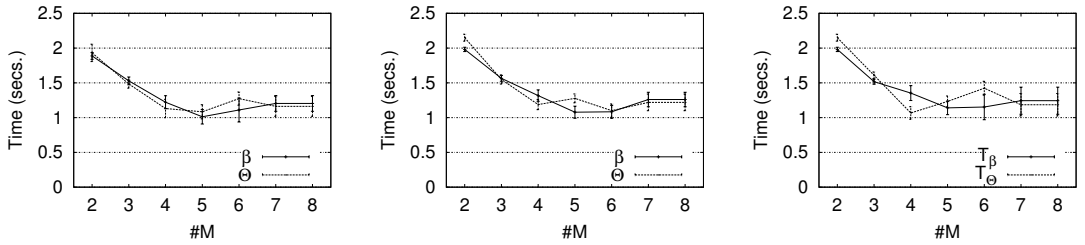
(a) W2D IB 512

(b) W2D TCP 512

Figure 7.4: Comparison of Wave2D measured communication times for partitions generated by Beaumont algorithm using the half perimeter metric β and the τ–Lop metric Θ using hierarchical heuristic. Measurements are for increasing number of nodes (and processes) on Infiniband and TCP networks and point-to-point communication mode. Matrix size is $N = 512$. Times are in milliseconds per iteration of the Wave2D simulation.

concurrently the *pbc* to the processes p_2 and p_6 running on node M_2. In this case, there is not much difference between metrics. This is cause the sender processes flood the destination node, degrading the performance because of contention, with no overall appreciable benefit over the half perimeter metric. Also, Figure 7.4 shows results of executing algorithm on the partitions generated using both half perimeters and performance model metrics. In both cases, point-to-point communication is the only pattern allowed. There is a significant performance improvement when the partition algorithm is guided by the τ–Lop metric, both for Infiniband and TCP networks.

Last set of performance figures are for matrices of size $N = 256$, shown in Figure 7.5 and Figure 7.6 for SUMMA kernel under Infiniband and TCP networks and different patterns, and Figure 7.7 for the Wave2D kernel on both networks. In these figures, we observed that for a higher communication volume in matrices of size the performance improvement for *ring* pattern in SUMMA is due to the usage of the hierarchical order, that reduces the network communication. Performance improvement of the Wave2D kernel is reduced with respect to Figure 7.4 because



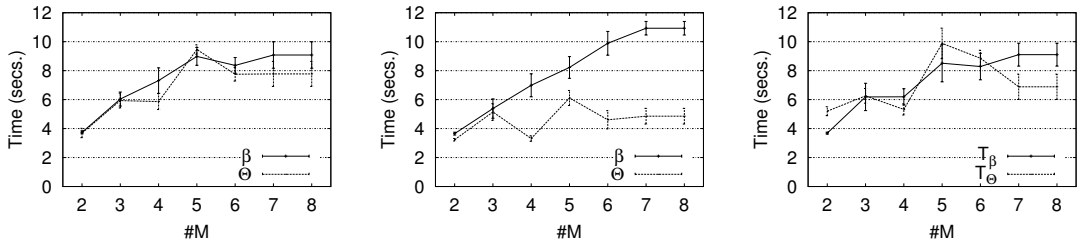
(a) SUMMA IB P2P 256 (b) SUMMA IB Ring 256 (c) SUMMA IB Bcast 256

Figure 7.5: Comparison of SUMMA measured communication times for partitions generated by Beaumont et al. algorithm using the half perimeter metric β and τ–Lop metric Θ using the hierarchical order. Measurements are for increasing number of nodes (and processes) on an Infiniband network and different communication modes (point-to-point, Ring and Broadcast). Matrices size is $N = 256$.

the message sizes interchanged for matrix of size $N = 256$ is as an average quarter than $N = 512$.

To calculate speedup, we use the formule β/Θ, which compare each metric. We show the maximum and average speedups for the set of processes and node numbers in table 7.3. For the SUMMA kernel, as expected, when it is executed using advanced communication patterns (*Ring*) the improvement is higher, due to the half perimeter metric does not match the communication pattern and it is oblivious of the arrangement of the rectangles in the partition. We can not appreciate improvements under Infiniband network. Wave2D kernel shows high improvements in any network and matrix size. Wave2D communication pattern, although proportional to the volume of communication, is influenced by the closeness of the rectangles assigned to processes in the same node.

As demonstrated, this method provides with enough information and details, including accurate estimations, for such development without HPC resource consumption.



(a) SUMMA TCP P2P 256 (b) SUMMA TCP Ring 256 (c) SUMMA TCP Bcast 256

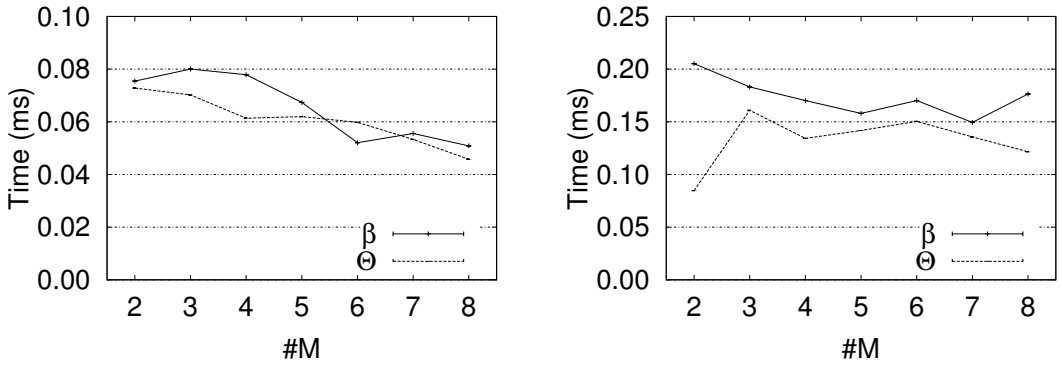
Figure 7.6: Comparison of SUMMA measured communication times for partitions generated by Beaumont et al. algorithm using the half perimeter metric β and τ–Lop metric Θ using the hierarchical order. Measurements are for increasing number of nodes (and processes) on a TCP network and different communication modes (point-to-point, Ring and Broadcast). Matrices size is $N = 256$. Times are in seconds of the complete execution of the kernel.

Table 7.2: Maximum and Average Experimental SUMMA Communication Speedups

	N	SUMMA IB P2P	SUMMA IB Ring	SUMMA IB Bcast	SUMMA TCP P2P	SUMMA TCP Ring	SUMMA TCP Bcast
Maximum	256	1,08	1.11	1.27	1.25	2.25	1.32
	512	1.29	1.25	1.23	1.63	1.77	1.33
Average	256	0.99	0.98	0.98	1.08	1.59	1.00
	512	0.98	0.96	0.98	1.08	1.41	0.92

Table 7.3: Maximum and Average Experimental W2D Communication Speedups

	N	W2D IB P2P	W2D TCP P2P
Maximum	256	1.27	2.42
	512	1.65	2.47
Average	256	1.07	1.28
	512	1.23	1.45



(a) W2D IB 256

(b) W2D TCP 256

Figure 7.7: Comparison of Wave2D measured communication times for partitions generated by Beaumont et al. algorithm using the half perimeter metric β and τ–Lop metric Θ using the hierarchical order. Measurements are for increasing number of nodes (and processes) on Infiniband and TCP networks and point-to-point communication mode. Matrix size is $N = 256$. Times are in milliseconds per iteration of the Wave2D simulation.

8

CONCLUSIONS

In this work, which is based in Rico-Gallego et al., 2018 and Rico-Gallego et al., 2015, we study the weaknesses of the partitioning algorithms and its metrics as Beaumont et al., 2001. We propose improvements for the measurement of the parameters, as well as the use as a metric for the partitioning of data and, finally, we study a hierarchical heuristic to improve communications.

Optimization of the communication in partitioning algorithms for data parallel kernels running on heterogeneous HPC platforms is needed. For this, non-uniform workload distribution between processes are proposed which is solved by using a geometrical volume-based metric to estimate the communication cost.

We demonstrate the importance of a correct measurement of parameters,achieving a fairly high efficiency index in this aspect, which will have a high impact on the importance of the tool in terms of time estimation based on communication.

We also present a new metric using analytical communication performance models to estimate the communication with a good accuracy. This metric estimates the communication costs of a certain kernel, in the case of this work, Wave2D and SUMMA, in a precise manner and improving the results obtained by half-perimeter metric. We also describe a tool which can automatize this process. This tool is quite novel since in this area there are no tools that contemplate these problems efficiently. It is based in τ–Lop model which can be used in partitioning algorithms to achieve higher performance partitions, better fitted to the platform details, as different networks and communication channels, and also to kernel features, as communication patterns.

The τ–Lop communication model encourage the development of better oriented heuristics (hierarchical order) to improve partitions for different HPC networks and kernels. Experimental results on a highly heterogeneous HPC platform confirm comparable performance of the SUMMA kernel to that of volume-based metrics on an Infiniband network, whereas they show a high improvement in a TCP network, as well as for Wave2D kernel in both networks. This is because, in TCP, communications have a high degree of impact, while Infiniband, being a high-speed network, the differences we achieve by grouping speeds by nodes and not by processes, are not so remarkable.

9

ACKNOWLEDGEMENTS

This work has been developed by the HPC Unex research group *http://hpc.unex.es/*. It was supported by the European Regional Development Fund 'A way to achieve Europe' (ERDF) and the Extremadura Local Government (Ref. IB16118) and by the computing facilities of Extremadura Research Center for Advanced Technologies (CETA-CIEMAT), funded by the European Regional Development Fund (ERDF).

REFERENCES

ALEXANDROV, A.; IONESCU, M. F.; SCHAUSER, K. E.; SCHEIMAN, C., 1995. LogGP: incorporating long messages into the LogP model - one step closer towards a realistic model for parallel computation. In: *LogGP: incorporating long messages into the LogP model - one step closer towards a realistic model for parallel computation. Proc. of the seventh annual ACM symposium on Parallel algorithms and architectures*. Santa Barbara, California, United States, pp. 95–105. SPAA '95. ISBN 0-89791-717-0.

BEAUMONT, O.; BOUDET, V.; RASTELLO, F.; ROBERT, Y., 2001. Matrix Multiplication on Heterogeneous Platforms. *IEEE Trans. Parallel Distrib. Syst.* Vol. 12, no. 10, pp. 1033–1051. ISSN 1045-9219. Available from DOI: 10.1109/71.963416.

BOSQUE, J.; PASTOR, L., 2006. A Parallel Computational Model for Heterogeneous Clusters. *IEEE Transactions on Parallel and Distributed Systems*. Vol. 17, pp. 1390–1400. ISSN 1045-9219. Available from DOI: doi.ieeecomputersociety.org/10.1109/TPDS.2006.165.

CULLER, D.; KARP, R.; PATTERSON, D.; SAHAY, A.; SCHAUSER, K. E.; SANTOS, E.; SUBRAMONIAN, R.; EICKEN, T. von, 1993. LogP: towards a realistic model of parallel computation. In: *LogP: towards a realistic model of parallel computation. Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*. San Diego, California, United States: ACM, pp. 1–12. PPOPP '93. ISBN 0-89791-589-5.

KIELMANN, T.; BAL, H. E.; VERSTOEP, K., 2000. Fast Measurement of LogP Parameters for Message Passing Platforms. In: *Fast Measurement of LogP Parameters for Message Passing Platforms. Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*. London, UK, UK: Springer-Verlag, pp. 1176–1183. IPDPS '00. ISBN 3-540-67442-X.

LASTOVETSKY, A.; MKWAWA, I.-H.; O'FLYNN, M., 2006. An accurate communication model of a heterogeneous cluster based on a switch-enabled Ethernet network. In: *An accurate communication model of a heterogeneous cluster based on a switch-enabled Ethernet network. Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on*. Vol. 2, 6 pp.–. ISSN 1521-9097.

LASTOVETSKY, A.; REDDY, R., 2007. Data Distribution for Dense Factorization on Computers with Memory Heterogeneity. *Parallel Comput.* Vol. 33, no. 12, pp. 757–779. ISSN 0167-8191.

RICO-GALLEGO, J. A.; LASTOVETSKY, A. L.; DÍAZ-MARTÍN, J. C., 2017. Model-Based Estimation of the Communication Cost of Hybrid Data-Parallel Applications on Heterogeneous Clusters. *IEEE Transactions on Parallel and Distributed Systems*. Vol. 28, no. 11, pp. 3215–3228. ISSN 1045-9219. Available from DOI: 10.1109/TPDS.2017.2715809.

RICO-GALLEGO, J. A.; DÍAZ-MARTÍN, J. C.; CALVO-JURADO, C.; MORENO-ÁLVAREZ, S.; GARCÍA-ZAPATA, J. L., 2018. Analytical Communication Performance Models as a metric in the partitioning of data-parallel kernels on heterogeneous platforms. *The Journal of Supercomputing*. ISSN 1573-0484. Available from DOI: 10.1007/s11227-018-2724-8.

RICO-GALLEGO, J.-A.; DÍAZ-MARTÍN, J.-C., 2015. τ-Lop: Modeling performance of shared memory MPI. *Parallel Computing*. Vol. 46, pp. 14 –31. ISSN 0167-8191. Available from DOI: https://doi.org/10.1016/j.parco.2015.02.006.

ZHONG, Z. ; RYCHKOV, V.; LASTOVETSKY, A., 2015. Data Partitioning on

Multicore and Multi-GPU Platforms Using Functional Performance Models. *IEEE Transactions on Computers*. Vol. 64, pp. 2506–2518. Available from DOI: 10.1109/TC.2014.2375202.