Optimal Matrix Partitioning for Data Parallel Computing on Hybrid Heterogeneous Platforms

1st Tania Malik
School of Computer Science (UCD)
Dublin, Ireland
tania.malik@ucd.ie

2nd Alexey Lastovetsky
School of Computer Science (UCD)
Dublin, Ireland
alexey.lastovetsky@ucd.ie

***Abstract*—In this paper, we study the problem of partitioning a matrix over a small number of interconnected heterogeneous processors. This problem is crucial for data parallel dense linear algebra and other applications with similar communication patterns on modern hybrid servers, integrating several heterogeneous compute devices such as CPUs, GPUs and other accelerators. The objective is to balance the load of the heterogeneous devices while minimising the communication cost. While the problem has been solved for the case of two processors, it is still open for three and more processors. The state-of-the-art solution for the case of three processors uses a communication cost function, which does not accurately account for the total amount of data moved between processors and therefore leaves the question of its global optimality open.**

In this work, we propose a cost function, which accurately represents the total amount of data moved between processors. Then, we formulate and solve the problem of optimal partitioning of a square computational domain, using this accurate communication cost function. Finally, we propose and implement an original experimental methodology for accurate measurement of the communication time of parallel applications on hybrid heterogeneous servers, integrating multi-core CPUs and various accelerators. We apply this methodology to experimental validation of our mathematical result.

***Index Terms*—Matrix multiplication, data partitioning, hybrid platforms, heterogeneous platforms, non-rectangular partitioning, optimal partitioning, communication optimization, data parallelism**

I. Introduction

The problem of partitioning a matrix into a set of submatrices over an arbitrary number of heterogeneous processors is crucial when considering dense linear algebra kernels and other applications with similar communication patterns on heterogeneous platforms. This problem has received increased attention in the last few years. One of the reasons is that hybrid heterogeneous platforms that include multi-core CPUs, GPUs and other accelerators are becoming mainstream in the bid to improve the performance and energy efficiency of parallel applications.

This optimization problem was first introduced in [1], [2], looking for a load-balanced partitioning of a matrix into a set of rectangular submatrices. The communication cost of the application was first introduced in the problem in [3], looking now for a load-balanced matrix partitioning that would also minimise the communication cost. The communication cost was defined in [3] as the sum of half-perimeters of the rectangular submatrices, motivated by the communication cost of one iteration of two-dimensional parallel matrix-matrix multiplication algorithms such as Cannon's and SUMMA [4]. Unlike the original problem [1], [2], the extended communication-aware problem was proved NP-complete [3]. A good number of approximate sub-optimal solutions have been proposed for this problem since its introduction.

While NP-complete in the general case of an arbitrary number of processors, the problem [3] has simple exact solutions for some specific small numbers of processors such as two and three. These solutions have a significant practical value for modern hybrid compute nodes that integrate a small number of heterogeneous compute devices such as CPUs, GPUs and other accelerators. However, the global optimality of these solutions was challenged in [5], [6].

In [5], the authors discovered a *non-rectangular* partitioning of a square matrix, the communication cost of which for parallel matrix-matrix multiplication on two heterogeneous processors was smaller than that of the *rectangular* partitioning, when the speed ratio of the processors was greater than $3:1$. Similarly, in [6], they designed a *non-rectangular* partitioning of a square matrix between three heterogeneous processors that was superior to any *rectangular* partitioning for some ratios of the speeds of the processors. These findings motivated research in optimal communication cost-aware matrix partitioning over a small number of heterogeneous processors, when no assumptions about shapes of optimal partitions are made, that is, considering any possible partition as potentially optimal.

In the case of two processors, it was mathematically proven in [7] that for any speed ratio between the processors either the rectangular partition or the non-rectangular partition discovered in [5] will always be optimal, that is, no other arbitrary partition will outperform them. However, the case of three processors appeared to be much harder. While six potentially optimal partition shapes were identified in [8], their optimality was not mathematically proven.

Most recently, Beaumont et al [9] proved the optimality of three partition shapes out of the six proposed in [8]. Their solution however does not use the total amount of data moved

This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant Number 14/IA/2474.

978-1-7281-8946-8/20/$31.00 ©2020 IEEE
DOI 10.1109/ISPDC51135.2020.00011

between processors to measure the communication cost of the partition. Instead, it introduces an approximate cost function, namely, the sum of half-perimeters (SHP) of rectangles, each covering the elements of the domain allocated to the same processor, and considers the partition optimal if it minimizes this cost function. This solution does not consider all possible partitions when proving the optimality of the shapes, thus leaving the question of their global optimality open. In this paper, we revisit the problem of optimal partitioning of a square computational domain between three interconnected heterogeneous processors, and solve the problem using the exact measure of the communication cost of the partition, proving thus the global optimality of the identified partition shapes.

The major contributions of this research work are:

1) We introduce a communication cost function for parallel computing on interconnected heterogeneous processors, which accurately represents the total amount of data moved between the processors.
2) We formulate and solve the problem of optimal partitioning of a square computational domain between three heterogeneous processors using this accurate communication cost function.
3) We propose and implement an original experimental methodology for accurate measurement of the communication time of parallel applications on hybrid heterogeneous platforms integrating multi-core CPUs and various accelerators. We apply this methodology to experimental validation of our solution.

The paper is organized as follows. Section II presents related work. Section III discusses the state-of-the-art solution of the matrix partitioning problem between for three processors. Section IV introduces the proposed exact cost communication function. Section V formulates the problem of optimal partitioning of a square computation domain between three heterogeneous processors, using the proposed cost function, and gives its solution. Section VI presents the experimental validation of the solution. Section VII concludes the paper and presents future research directions.

II. Related Work

The performance of a data-parallel application on a heterogeneous platform is mainly determined by the partitioning of its computational domain between the heterogeneous processes. The optimal partitioning would minimise the execution time of the application.

Matrices and other rectangular computational domains are omnipresent in computational science. The problem of partitioning a matrix into set of submatrices was first introduced in [1], [2]. The heterogeneous processors were modelled by their constant relative speeds, and the objective was to partition the matrix into rectangular submatrices so that to minimise the computation time. Due to simplicity of the performance model of the heterogeneous platform, this problem had simple exact solutions and efficient algorithms finding these solutions. A more realistic variant of this problem, using a smooth functional performance model of heterogeneous processors, where the speed of each processor was represented by a smooth function of the problem size, was introduced in [10] and studied in [10], [11], [12], [13], [14], [15]. The important artefact of these optimisation problems is that any optimal solution would balance the load of the heterogeneous processors.

With the advent of multicore processors, the assumption of the smoothness of the speed function of a processor became less realistic [16], [17], and a new variant of the problem, using arbitrary discrete speed functions of the processors, was introduced in [17] and studied in [18]. Unlike its predecessors, optimal solutions of this problem does not have to balance the load of the processors.

The execution time of a data parallel application includes both the computation time and the communication time. The communication cost was first introduced in the matrix partitioning problem in [3], extending the original problem [1] by looking now not just for a load-balanced matrix partitioning but for the load-balance partitioning, also minimising the communication cost. The communication cost was defined in [3] as the sum of half-perimeters of the rectangular submatrices and was motivated by the communication cost of one iteration of two-dimensional parallel matrix-matrix multiplication algorithms such as Cannon's and SUMMA [4]. Unlike the original problem of [1], the extended communication-aware problem was proved NP-complete [3]. The first approximation algorithm with a bounded ratio of 1.75 was also proposed in [3]. A good number of other approximate sub-optimal solutions have been proposed for this problem and its variants that use the smooth and arbitrary discrete speed functions, such as [19], [20], [21], [22], [23], [24].

In [5], it was discovered that non-rectangular partitions can outperform rectangular. For parallel multiplication of square matrices on two heterogeneous processors, the authors show that if the ratio of the processors' speeds is less than $1:3$, then allocation of a square area in the top left corner of the matrix to a slower processor, and the balance of the matrix to the faster processor, will result in a lower number of matrix elements moved between the processors in comparison with the straightforward rectangular partitioning.

For the same application, a *non-rectangular* partitioning of a square matrix between three heterogeneous processors was designed in [6] that appeared superior to any *rectangular* partitioning for some ratios of the speeds of the processors. This non-rectangular partitioning allocates squares in the opposite corners of the matrix to two slower processors, and the balance to the faster one.

These findings motivated research in optimal communication-aware matrix partitioning over heterogeneous processors, when no assumptions about the shapes of optimal partitions are made. In the case of two processors, an original mathematical method, called the Push technique, was developed in [7] to mathematically prove that for any speed ratio between the processors either the rectangular partition or the non-rectangular partition discovered in [5] will always be optimal, that is, no other arbitrary partition will outperform

them. In [8] and [25], the Push technique was extended and applied to the case of three heterogeneous processors. While it helped identify six potentially optimal shapes, three of which were non-rectangular, their optimality was not mathematically proven. In [26], by relaxing the restriction of rectangular partitioning, a recursive approximation matrix partitioning algorithm for an arbitrary number of processors managed to reduce the approximation ratio to $2/\sqrt{3}$, which is the best known approximation.

Most recently, Beaumont et al [9] revisit the communication-aware matrix partitioning problem for three processors. They formulate and mathematically solve the problem using a cost function, which has some relation to the amount of data moved between the processors. This way they mathematically prove the optimality of three particular partition shapes and analyze the accuracy of best approximate solutions against the optimal solutions. The communication cost function of a partition used in this work is motivated by two-dimensional data parallel matrix multiplication algorithms, such as Cannon's and SUMMA, and is calculated as the sum of half-perimeters of rectangles, each covering the elements of the matrix allocated to the same processor. While approximating well the total amount of data moved between processors during parallel matrix multiplication in the case of rectangular partitions, this cost function is inaccurate in the genaral case of arbitrary partition, not able to discriminate many partitions, which in practice have different communication costs. Therefore, this solution does not consider all possible partitions when proving the optimality of the identified shapes, leaving the question of their global optimality open.

In order to accurately measure the actual communication cost, we need a cost function, which accounts for all matrix elements moved between processors. Such a cost function has been proposed in [7], counting the total number of communicated matrix elements. However, it proved to be hard to mathematically prove the optimality of the identified partition shapes using this discrete metric. In our work, we overcome this difficulty by extending this discrete cost function into the continuous space and proposing an exact real-valued communication cost function for an arbitrary partition of a real-valued square. Like the cost function of [9], our cost function is also motivated by two-dimensional data-parallel matrix multiplication applications [3], [27], [21]. These applications compute the product $C = A \times B$ of two matrices A and B, where elements of matrices A, B and C are identically partitioned between processors in proportion to their relative speeds. Element c_{ij} is calculated as the dot product of i-th row of matrix A, A_i, and j-th column of matrix B, B_j. To calculate c_{ij}, all elements of A_i and B_j, which do not belong to the processor that owns c_{ij}, must be sent to this processor. Derived from this, our cost function would accurately and absolutely exactly reflect the total amount of data moved between the processors.

III. Optimal Partitioning a Square between Three Heterogeneous Processors: State of the Art

In this section, we revisit the problem of optimal partitioning of a square computational domain between three heterogeneous processors, which would partition the domain in proportion to the speed of the processors (in order to balance their load) and simultaneously minimize the total amount of data moved between the processors. The state-of-the-art solution of this problem proves that the three shapes shown in Fig. 1 will be sufficient to optimally partition a square [9].

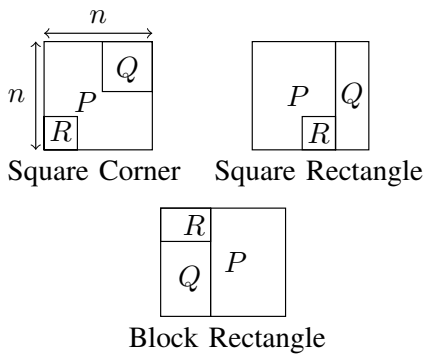


Fig. 1: Optimal partition shapes for a square computational domain for three heterogeneous processors. The Square Corner partition gives processors Q and R square regions. The Square Rectangle partition gives processor R a square region.

However, this solution of [9] does not use the total amount of data moved between processors to measure the cost of each partition. Instead, it introduces an approximate cost function, namely, the sum of half-perimeters (SHP) of rectangles, each covering the elements of the domain allocated to the same processor, and considers the partition optimal if it minimizes this cost function. This approximate cost function is derived from data parallel matrix multiplication algorithms and accurately represents their communication cost when the matrices are partitioned into rectangles. At the same time, the cost function of [9] will not discriminate many general partitions with different total amounts of data moved between the processors, resulting in the same SHP cost for them, as shown in Fig. 2, and on the other hand can discriminate partitions characterized by the same total amount of moved data as shown in Fig. 3. In these figures, the exact communication cost C of a given partition is derived as the total number of elements of matrices A and B moved between processors P, Q, and R during their parallel matrix-matrix multiplication, given the matrices are identically partitioned between the processors. S_X designates the area of the region marked X (that is, the total number of elements in this region). Thus, the solution of [9] does not consider all possible partitions when proving the optimality of the shapes in Fig. 1, leaving the question of their global optimality open.

In this work, we solve the problem of optimal partitioning using the exact measure of the communication cost of the

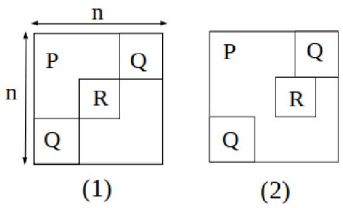
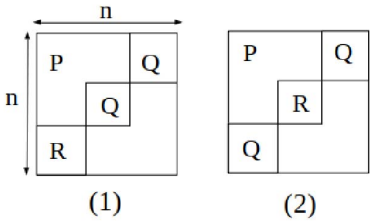
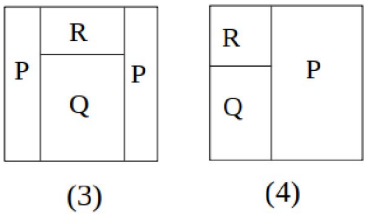


Fig. 2: Two partitions of a square $n \times n$ matrix between three heterogeneous processors P, Q, and R, with the same SHP cost and different exact communication costs C(1) and C(2). Given the regions marked by Q and R are all same size squares of sizes $(\frac{n}{3})$, we have: $SHP(1) = SHP(2) = 4n + 2\sqrt{S_R}$, however, $C(1) = 2n^2 < C(2) = 2n^2 - \frac{1}{2}\sqrt{S_R} \times n + \sqrt{S_Q} \times n$.



$SHP(1) = 2n + 4\sqrt{S_Q} + 2\sqrt{S_R} < SHP(2) = 4n + 2\sqrt{S_R}$
however, $C(1) = C(2) = 2n^2$



$SHP(3) = 3n + 2n - \frac{S_P}{n} > SHP(4) = 2n + 2n - \frac{S_P}{n}$
however, $C(3) = C(4) = 2n^2 - S_P$

Fig. 3: Two pairs of partitions, (1) and (2), (3) and (4), discriminated by the SHP cost function but having the same exact communication cost C.

partition, thus proving the global optimality of the identified partition shapes.

IV. Cost Function

In this section, we mathematically define the exact cost communication function of an arbitrary partition of a square $n \times n$ computational domain, assuming the same cost of communication of one data unit between different pairs of processors.

A. Definition of Discrete Cost Function

In this subsection, we derive a discrete cost function from the communication cost of parallel matrix-matrix multiplication. $C = A \times B$, of two square matrices A and B, assuming that the elements of matrices A, B and C are identically partitioned between processors in proportion with relative speeds of the processors. Element c_{ij} is calculated as the dot product of i-th row of matrix A, A_i, and j-th column of matrix B, B_j. To calculate c_{ij}, all elements of A_i and B_j, which do not belong to the processor that owns c_{ij}, must be sent to this processor. Derived from this observation, we define the communication cost of a partition of a square matrix to be equal to the total number of elements of matrices A and B moved between the processors.

Mathematically, each partition of an $n \times n$ matrix between three processors P, Q, and R, is represented by a mapping $[0, 1, .., n] \times [0, 1, .., n] \mapsto \{P, Q, R\}$, of the set of indices of the matrix into the set of processors. The set of all possible partitions is denoted as

$$\mathbb{M} = \left\{ [1, .., n] \times [1, .., n] \mapsto \{P, Q, R\} \right\},$$

and for each partition $M \in \mathbb{M}$ our cost function $\overline{f}_c : \mathbb{M} \mapsto \mathbb{Z}_{\geq 0}$ returns the defined communication cost, $\overline{f}_c(M)$.

Rows	—	Columns
$\overline{a}_{\mathbf{P}}$: Number of the rows where all elements of a row are allocated to a single processor P		$\overline{b}_{\mathbf{P}}$: Number of the columns where all elements of a column are allocated to a single processor P
$\overline{a}_{\mathbf{Q}}$: Number of the rows where all elements of a row are allocated to a single processor Q		$\overline{b}_{\mathbf{Q}}$: Number of the columns where all elements of a column are allocated to a single processor Q
$\overline{a}_{\mathbf{R}}$: Number of the rows where all elements of a row are allocated to a single processor R		$\overline{b}_{\mathbf{R}}$: Number of the columns where all elements of a column are allocated to a single processor R
$\overline{a}_{\mathbf{PQ}}$: Number of the rows where elements of a row are allocated among two processors P and Q		$\overline{b}_{\mathbf{PQ}}$: Number of the columns where elements of a column are allocated among two processors P and Q
$\overline{a}_{\mathbf{PR}}$: Number of the rows where elements of a row are allocated among two processors P and R		$\overline{b}_{\mathbf{PR}}$: Number of the columns where elements of a column are allocated among two processors P and R
$\overline{a}_{\mathbf{QR}}$: Number of the rows where elements of a row are allocated among two processors Q and R		$\overline{b}_{\mathbf{QR}}$: Number of the columns where elements of a column are allocated among two processors Q and R
$\overline{a}_{\mathbf{PQR}}$: Number of the rows where elements of row are allocated among all three processors P, Q and R		$\overline{b}_{\mathbf{PQR}}$ Number of the columns where elements of a column are allocated among all three processors P, Q and R
$\Sigma\overline{a} = n$		$\Sigma\overline{b} = n$

TABLE I: Parameters for the discrete cost function.

B. Analytical Formulas for Discrete Cost Function

In this subsection, we derive two analytical formulas for the cost $\overline{f}_c(M)$ of an arbitrary partition $M \in \mathbb{M}$, characterized by the parameters summarized in Table I.

Given $X \in \{P, Q, R\}$, let $\overline{A}_{PQR,X}$ be the total number of elements allocated to processor X in the $\overline{a}_{\text{PQR}}$ rows, elements of which are distributed between all three processors P, Q, and R. Similarly, let $\overline{B}_{PQR,X}$ be the total number of elements allocated to processor X in the $\overline{b}_{\text{PQR}}$ columns, elements of

which are distributed between all three processors P, Q, and R. Then, the total number of elements moved between any pair of processors $X, Y \in \{P, Q, R\}$ can be calculated as follows:

$$C_{XY} = (\overline{a}_{XY} \times n) + (n \times \overline{b}_{XY}) + (\overline{A}_{PQR,X} + \overline{A}_{PQR,Y}) + (\overline{B}_{PQR,X} + \overline{B}_{PQR,Y})$$

Note that
$(\overline{A}_{PQR,P} + \overline{A}_{PQR,Q} + \overline{A}_{PQR,R}) = \overline{a}_{PQR} \times n$, and
$(\overline{B}_{PQR,P} + \overline{B}_{PQR,Q} + \overline{B}_{PQR,R}) = \overline{b}_{PQR} \times n$.

Therefore,

$$\begin{aligned}
\overline{f}_c(M) &= C_{PQ} + C_{PR} + C_{QR} \\
&= (\overline{a}_{PQ} + \overline{b}_{PQ} + \overline{a}_{PR} + \overline{b}_{PR} + \overline{a}_{QR} + \overline{b}_{QR}) \times n \\
&+ \overline{A}_{PQR,P} + \overline{A}_{PQR,Q} + \overline{B}_{PQR,P} + \overline{B}_{PQR,Q} \\
&+ \overline{A}_{PQR,P} + \overline{A}_{PQR,R} + \overline{B}_{PQR,P} + \overline{B}_{PQR,R} \\
&+ \overline{A}_{PQR,Q} + \overline{A}_{PQR,R} + \overline{B}_{PQR,Q} + \overline{B}_{PQR,R} \\
&= (\overline{a}_{PQ} + \overline{b}_{PQ} + \overline{a}_{PR} + \overline{b}_{PR} + \overline{a}_{QR} + \overline{b}_{QR}) \times n \\
&+ 2 \times (\overline{A}_{PQR,P} + \overline{A}_{PQR,Q} + \overline{A}_{PQR,R}) \\
&+ 2 \times (\overline{B}_{PQR,P} + \overline{B}_{PQR,Q} + \overline{B}_{PQR,R}) \\
&= (\overline{a}_{PQ} + \overline{b}_{PQ} + \overline{a}_{PR} + \overline{b}_{PR} + \overline{a}_{QR} + \overline{b}_{QR} \\
&\quad + 2 \times \overline{a}_{PQR} + 2 \times \overline{b}_{PQR}) \times n
\end{aligned} \tag{1}$$

As
$(\overline{a}_P + \overline{a}_Q + \overline{a}_R + \overline{a}_{PQ} + \overline{a}_{PR} + \overline{a}_{QR} + \overline{a}_{PQR}) \times n = n \times n = n^2$,
$(\overline{b}_P + \overline{b}_Q + \overline{b}_R + \overline{b}_{PQ} + \overline{b}_{PR} + \overline{b}_{QR} + \overline{b}_{PQR}) \times n = n \times n = n^2$,
the alternative formula will be

$$\begin{aligned}
\overline{f}_c(M) &= 2 \times n^2 - (\overline{a}_P + \overline{a}_Q + \overline{a}_R + \overline{b}_P + \overline{b}_Q + \overline{b}_R) \times n \\
&\quad + (\overline{a}_{PQR} + \overline{b}_{PQR}) \times n
\end{aligned} \tag{2}$$

C. Continuous Extension of Discrete Cost Function

While motivated by the problem of optimal partitioning of a square $n \times n$ matrix between three heterogeneous processors, in this work we aim to solve a more general problem, namely, the problem of optimal partitioning of a real-valued $[0, n] \times [0, n]$ square. Each partition T of the $[0, n] \times [0, n]$ square between three processors P, Q, and R, is defined as a mapping $T : [0, n] \times [0, n] \mapsto \{P, Q, R\}$ such that the inverse images $T^{-1}(P)$,$T^{-1}(Q)$, and $T^{-1}(R)$, are all Lebesgue-Borel measurable sets; the measure of the Lebesgue-Borel measurable set L is here denoted by $\mu(L)$. The set of all possible partitions is denoted as

$$\mathbb{T} = \left\{ [0, n] \times [0, n] \mapsto \{P, Q, R\} \right\},$$

and each partition $T \in \mathbb{T}$ is characterized by the parameters summarized in Table II. Note that if we consider the $[0, n] \times [0, n]$ square as a $n \times n$ set of unit squares, that is, squares of size 1×1, then any partition $T \in \mathbb{T}$, which is mapping each unit square to a single processor, will represent a matrix partition, $M \in \mathbb{M}$.

Now for each partition $T \in \mathbb{T}$, we define the cost function $f_c(T)$ as follows

$$\begin{aligned}
f_c(T) = (&a_{PQ} + b_{PQ} + a_{PR} + b_{PR} + a_{QR} + b_{QR} \\
&+ 2 \times a_{PQR} + 2 \times b_{PQR}) \times n
\end{aligned} \tag{3}$$

This definition guarantees that if $T \in \mathbb{T}$ represents the matrix partition $M \in \mathbb{M}$, then $f_c(T) = \overline{f}_c(M)$.

Also, as
$(a_P + a_Q + a_R + a_{PQ} + a_{PR} + a_{QR} + a_{PQR}) \times n = n \times n = n^2$,
and
$(b_P + b_Q + b_R + b_{PQ} + b_{PR} + b_{QR} + b_{PQR}) \times n = n \times n = n^2$,
the alternative formula will be

$$\begin{aligned}
f_c(T) = 2 \times n^2 &- (a_P + a_Q + a_R + b_P + b_Q + b_R) \times n \\
&+ (a_{PQR} + b_{PQR}) \times n
\end{aligned} \tag{4}$$

Horizontal lines	—	Vertical lines
$a_{\mathbf{P}}$: Measure of the set of the horizontal lines where all points of a line are mapped to processor P		$b_{\mathbf{P}}$: Measure of the set of the vertical lines where all points of a line are mapped to processor P
$a_{\mathbf{Q}}$: Measure of the set of the horizontal lines where all points of a line are mapped to processor Q		$b_{\mathbf{Q}}$: Measure of the set of the vertical lines where all points of a line are mapped to processor Q
$a_{\mathbf{R}}$: Measure of the set of the horizontal lines where all points of a line are mapped to processor R		$b_{\mathbf{R}}$: Measure of the set of the vertical lines where all points of a line are mapped to processor R
$a_{\mathbf{PQ}}$: Measure of the set of the horizontal lines where points of a line are mapped among two processors P and Q		$b_{\mathbf{PQ}}$: Measure of the set of the vertical lines where points of a line are mapped among two processors P and Q
$a_{\mathbf{PR}}$: Measure of the set of the horizontal lines where points of a line are mapped among two processors P and R		$b_{\mathbf{PR}}$: Measure of the set of the vertical lines where points of a line are mapped among two processors P and R
$a_{\mathbf{QR}}$: Measure of the set of the horizontal lines where points of a line are mapped among two processors Q and R		$b_{\mathbf{QR}}$: Measure of the set of the vertical lines where points of a line are mapped among two processors Q and R
$a_{\mathbf{PQR}}$: Measure of the set of the horizontal lines where points of line are mapped among all three processors P, Q and R		$b_{\mathbf{PQR}}$: Measure of the set of the vertical lines where points of a line are mapped among all three processors P, Q and R
$\Sigma a = n$		$\Sigma b = n$

TABLE II: Parameters for the real valued cost function.

V. Optimal Partitions of Square

Now we formulate the problem of optimal partitioning of a square with the cost function $f_c(T)$ defined in the previous subsection and give its solution.

Problem: Given a real-valued square $[0, n] \times [0, n]$ and three positive real numbers $\{S_P, S_Q, S_R\}$ such that

$S_P + S_Q + S_R = n^2$, find a partition $T \in \mathbb{T}$ of this square that minimizes the cost function $f_c(T)$ and $S_P = \mu(T^{-1}(P))$, $S_Q = \mu(T^{-1}(Q))$, $S_R = \mu(T^{-1}(R))$.

A. Optimal Partition Shapes

Our main result is that one of the three partition shapes shown in Fig. 1 will always be a solution to this problem. To formulate this result mathematically and prove it, we first derive the cost of these three partitions. Let T_{SC} denote the Square Corner partition, T_{SR} denote the Square Rectangle partition, and T_{BR} denote the Block Rectangle partition. Note, that in the case of the considered partitions, the measure of each of the regions of the square mapped to a single processor P, Q, or R, will be equivalent to the normal area of this region and will be equal to S_P, S_Q, S_R respectively. Then the following three lemmas give us formulas for the cost of these partitions.

Lemma 1. $f_c(T_{SC}) = 2 \times n \times (\sqrt{S_R} + \sqrt{S_Q})$.

Proof. Here, $a_{PQR} = b_{PQR} = a_{QR} = b_{QR} = 0$.
On the other hand, $a_{PQ} = \sqrt{S_Q} \times n$, $a_{PR} = \sqrt{S_R} \times n$ and $b_{PQ} = n \times \sqrt{S_Q}$ and $b_{PR} = n \times \sqrt{S_R}$. Therefore, according to formula (3):

$$f_c(T_{SC}) = 2 \times n \times (\sqrt{S_R} + \sqrt{S_Q}) \tag{5}$$

□

Lemma 2. $f_c(T_{SR}) = n^2 + 2 \times (\sqrt{S_R} \times n)$.

Proof. Here, $a_{PQR} = \sqrt{S_R} \times n$, $a_{QR} = a_{PR} = 0$, $a_{PQ} = (n - \sqrt{S_R}) \times n$, $b_{PQR} = b_{PQ} = b_{QR} = 0$, and $b_{PR} = n \times \sqrt{S_R}$. Thus, according to formula (3):

$$f_c(T_{SR}) = n^2 + 2 \times (\sqrt{S_R} \times n) \tag{6}$$

□

Lemma 3. $f_c(T_{BR}) = 2 \times n^2 - S_P$.

Proof. Here, $a_{PQR} = a_P = a_Q = a_R = 0$, $b_{PQR} = b_Q = b_R = 0$ and $n \times b_P = S_P$. Therefore, according to formula (4):

$$f_c(T_{BR}) = 2 \times n^2 - b_P \times n = 2 \times n^2 - S_P \tag{7}$$

□

The following theorem formulates our main result for Problem 1.

Theorem 1. $\forall T \in \mathbb{T} : (f_c(T) \geq f_c(T_{SC})) \vee (f_c(T) \geq f_c(T_{SR})) \vee (f_c(T) \geq f_c(T_{BR}))$.

Proof. The proof is split into lemmas, depending on the measure of the set of horizontal and vertical lines where all points of a line are mapped to a single processor P, Q and R for any partition T.

Lemma 1.1. *If* $(a_P = 0) \wedge (a_Q = 0) \wedge (a_R = 0) \wedge (b_P = 0) \wedge (b_Q = 0) \wedge (b_R = 0)$*, then* $f_c(T) \geq f_c(T_{BR})$.

Proof. According to formula (4):

$$\begin{aligned} f_c(T) &= 2 \times n^2 + (a_{PQR} \times n) + (n \times b_{PQR}) \\ &\geq 2 \times n^2 > 2 \times n^2 - S_P = f_c(T_{BR}) \end{aligned}$$

□

Lemma 1.2. *If* $(a_P > 0) \oplus (a_Q > 0) \oplus (a_R > 0) \oplus (b_P > 0) \oplus (b_Q > 0) \oplus (b_R > 0)$*, then* $f_c(T) \geq f_c(T_{BR})$.

Proof. Let there exist exactly one $X \in \{P, Q, R\}$ such that $a_X > 0$ and $b_P = b_Q = b_R = 0$. Then, according to formula (4), $f_c(T) = 2 \times n^2 - (a_X \times n) + (a_{PQR} \times n) + (n \times b_{PQR})$, where $a_X \times n \leq S_X$, so that $f_c(T) \geq 2 \times n^2 - S_X$.
As $S_P \geq S_Q \geq S_R$, $S_X \leq \max\{S_P, S_Q, S_R\} = S_P$. Therefore, $f_c(T) \geq 2 \times n^2 - S_X \geq 2 \times n^2 - S_P = f_c(T_{BR})$.
Similarly, $f_c(T) \geq f_c(T_{BR})$ when there exists exactly one $X \in \{P, Q, R\}$ such that $b_X > 0$ and $a_P = a_Q = a_R = 0$.

□

Lemma 1.3. *If* $[(a_P > 0) \wedge (b_P > 0)] \oplus [(a_Q > 0) \wedge (b_Q > 0)] \oplus [(a_R > 0) \wedge (b_R > 0)]$ *, then* $f_c(T) \geq f_c(T_{SC})$.

Proof. In this case, we assume that there is exactly one processor $X \in \{P, Q, R\}$ such that $a_X > 0$ and $b_X > 0$ and for remaining processors Y and Z, $a_Y = a_Z = b_Y = b_Z = 0$. Also, as any horizontal line and any vertical line contain a point mapped to X, therefore, $a_{YZ} = b_{YZ} = 0$.
Then, according to formula (3),
$f_c(T) = (a_{XY} + b_{XY} + a_{XZ} + b_{XZ} + 2 \times a_{XYZ} + 2 \times b_{XYZ}) \times n$.
The measure of the set of all horizontal and vertical lines containing points mapped to Y will be equal to $(a_{XY} + a_{XYZ} + b_{XY} + b_{XYZ})$ and cannot be less than the half-perimeter of a square with the area of S_Y. Therefore, $a_{XY} + a_{XYZ} + b_{XY} + b_{XYZ} \geq 2 \times \sqrt{S_Y}$. Similarly, $a_{XZ} + a_{XYZ} + b_{XZ} + b_{XYZ} \geq 2 \times \sqrt{S_Z}$.
Thus, $f_c(T) =$
$(a_{XY} + b_{XY} + a_{XZ} + b_{XZ} + 2 \times a_{XYZ} + 2 \times b_{XYZ}) \times n =$
$((a_{XY} + a_{XYZ} + b_{XY} + b_{XYZ}) +$
$(a_{XZ} + a_{XYZ} + b_{XZ} + b_{XYZ})) \times n \geq$
$2 \times (\sqrt{S_Y} + \sqrt{S_Z}) \times n \geq 2 \times (\sqrt{S_Q} + \sqrt{S_R}) \times n = f_c(T_{SC})$.

□

Lemma 1.4. *If* $[(a_P > 0) \wedge (a_Q > 0)] \oplus [(a_P > 0) \wedge (a_R > 0)] \oplus [(a_Q > 0) \wedge (a_R > 0)] \oplus [(b_P > 0) \wedge (b_Q > 0)] \oplus [(b_P > 0) \wedge (b_R > 0)] \oplus [(b_Q > 0) \wedge (b_R > 0)]$ *, then* $f_c(T) \geq f_c(T_{SR})$

Proof. Let $X, Y \in \{P, Q, R\}$, $a_X > 0$ and $a_Y > 0$ while $b_X = b_Y = 0$. Note that for the remaining processor $Z \in \{P, Q, R\}$, $a_Z = b_Z = 0$, Then, according to formula (3), $f_c(T) = (a_{XY} + a_{XZ} + a_{YZ} + b_{XY} + b_{XZ} + b_{YZ} + 2(a_{XYZ} + b_{XYZ})) \times n$.

The measure of the set of all horizontal and vertical lines containing points mapped to Z will be equal to $(a_{XZ} + a_{YZ} + a_{XYZ} + b_{XZ} + b_{YZ} + b_{XYZ})$ and cannot be less than the half-perimeter of a square with the area of S_Z. Therefore,

$a_{XZ} + a_{YZ} + a_{XYZ} + b_{XZ} + b_{YZ} + b_{XYZ} \geq 2 \times \sqrt{S_Z}$. Also, all vertical lines contain points from X and Y, therefore, $b_{XY} + b_{XYZ} = n$. Thus,

$$\begin{aligned} f_c(T) &= (a_{XY} + a_{XZ} + a_{YZ} + b_{XY} + b_{XZ} + b_{YZ} + \\ &\quad 2(a_{XYZ} + b_{XYZ})) \times n \\ &\geq (a_{XY} + a_{XYZ} + b_{XY} + b_{XYZ}) \times n + 2(\sqrt{S_Z} \times n) \\ &\geq n \times (b_{XY} + b_{XYZ}) + 2(\sqrt{S_Z} \times n) \\ &\geq n^2 + 2(\sqrt{S_Z} \times n) \\ &\geq n^2 + 2(\sqrt{S_R} \times n) = f_c(T_{SR}). \end{aligned}$$

Similarly, $f_c(T) \geq f_c(T_{SR})$ when there exist X, $Y \in \{P, Q, R\}$ such that $b_X > 0$ and $b_Y > 0$.

□

Lemma 1.5. *If* $[(a_P > 0) \wedge (a_Q > 0) \wedge (a_R > 0)] \oplus [(b_P > 0) \wedge (b_Q > 0) \wedge (b_R > 0)]$ *, then* $f_c(T) \geq f_c(T_{BR})$

Proof. Let $(a_\mathrm{P} > 0) \wedge (a_\mathrm{Q} > 0) \wedge (a_\mathrm{R} > 0)$. Then, $b_\mathrm{P} = b_\mathrm{Q} = b_\mathrm{R} = 0$ and $b_{PQR} = n \times n = n^2$. Therefore, according to formula (4),

$$\begin{aligned} f_c(T) &\geq 2 \times n^2 - ((a_\mathrm{P} + a_\mathrm{Q} + a_\mathrm{R}) \times n) + (a_{PQR} \times n) + n^2 \\ &\geq 2 \times n^2 \\ &> 2 \times n^2 - S_P = f_c(T_{BR}). \end{aligned}$$

Similarly, $f_c(T) \geq f_c(T_{BR})$ when $(b_\mathrm{P} > 0) \wedge (b_\mathrm{Q} > 0) \wedge (b_\mathrm{R} > 0)$. □

Thus, we have proved that for any feasible combination of the partition parameters from Table II, there is always an optimal solution to Problem 1 with a shape from Fig. 1. The last lemma in this section proves that all possible combinations of the partition parameters are covered by lemmas 1.1 – 1.5.

Lemma 1.6. *For any partition $T \in \mathbb{T}$, its parameters from Table II will satisfy at least one of the cases of lemmas 1.1 – 1.5.*

Proof. Let us denote $\mathfrak{A}_\mathrm{P} \equiv (a_\mathrm{P} > 0), \mathfrak{A}_\mathrm{Q} \equiv (a_\mathrm{Q} > 0), \mathfrak{A}_\mathrm{R} \equiv (a_\mathrm{R} > 0), \mathfrak{B}_\mathrm{P} \equiv (b_\mathrm{P} > 0), \mathfrak{B}_\mathrm{Q} \equiv (b_\mathrm{Q} > 0), \mathfrak{B}_\mathrm{R} \equiv (b_\mathrm{R} > 0)$.

Then, the case of lemma 1.1 can be expressed as follows,
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R})$

The case of lemma 1.2:
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \mathfrak{B}_\mathrm{R}) \oplus$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \mathfrak{B}_\mathrm{Q}) \oplus$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \mathfrak{B}_\mathrm{P}) \oplus$
$(\neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \mathfrak{A}_\mathrm{R}) \oplus$
$(\neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{Z} \wedge \neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \mathfrak{A}_\mathrm{Q}) \oplus$
$(\neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \mathfrak{A}_\mathrm{P})$

The case of lemma 1.3:
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \mathfrak{A}_\mathrm{R} \wedge \mathfrak{B}_\mathrm{R}) \oplus$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \mathfrak{A}_\mathrm{Q} \wedge \mathfrak{B}_\mathrm{Q}) \oplus$
$(\neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \mathfrak{A}_\mathrm{P} \wedge \mathfrak{B}_\mathrm{P})$

The case of lemma 1.4:
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \mathfrak{B}_\mathrm{Q} \wedge \mathfrak{B}_\mathrm{R}) \oplus$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \mathfrak{B}_\mathrm{P} \wedge \mathfrak{B}_\mathrm{R}) \oplus$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \mathfrak{B}_\mathrm{P} \wedge \mathfrak{B}_\mathrm{Q}) \oplus$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \mathfrak{A}_\mathrm{Q} \wedge \mathfrak{A}_\mathrm{R}) \oplus$
$(\neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \mathfrak{A}_\mathrm{P} \wedge \mathfrak{A}_\mathrm{R}) \oplus$
$(\neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \mathfrak{A}_\mathrm{P} \wedge \mathfrak{A}_\mathrm{Q})$

The case of lemma 1.5:
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \mathfrak{B}_\mathrm{P} \wedge \mathfrak{B}_\mathrm{Q} \wedge \mathfrak{B}_\mathrm{R}) \oplus$
$(\mathfrak{A}_\mathrm{P} \wedge \mathfrak{A}_\mathrm{Q} \wedge \mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R})$

$[(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R})] \vee$
$[(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \mathfrak{B}_\mathrm{R}) \oplus$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \mathfrak{B}_\mathrm{Q}) \oplus$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \mathfrak{B}_\mathrm{P}) \oplus$
$(\neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \mathfrak{A}_\mathrm{R}) \oplus$
$(\neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \mathfrak{A}_\mathrm{Q}) \oplus$
$(\neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \mathfrak{A}_\mathrm{P})] \vee$
$[(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P}\neg\mathfrak{B}_\mathrm{Q} \wedge \mathfrak{A}_\mathrm{R} \wedge \mathfrak{B}_\mathrm{R}) \oplus$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \mathfrak{A}_\mathrm{Q} \wedge \mathfrak{B}_\mathrm{Q}) \oplus$
$(\neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \mathfrak{A}_\mathrm{P} \wedge \mathfrak{B}_\mathrm{P})] \vee$
$[(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \mathfrak{B}_\mathrm{Q} \wedge \mathfrak{B}_\mathrm{R}) \oplus$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \mathfrak{B}_\mathrm{P} \wedge \mathfrak{B}_\mathrm{R}) \oplus$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \mathfrak{B}_\mathrm{P} \wedge \mathfrak{B}_\mathrm{Q}) \oplus$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \mathfrak{A}_\mathrm{Q} \wedge \mathfrak{A}_\mathrm{R}) \oplus$
$(\neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \mathfrak{A}_\mathrm{P} \wedge \mathfrak{A}_\mathrm{R}) \oplus$
$(\neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R} \wedge \mathfrak{A}_\mathrm{P} \wedge \mathfrak{A}_\mathrm{Q})] \vee$
$[(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \mathfrak{B}_\mathrm{P} \wedge \mathfrak{B}_\mathrm{Q} \wedge \mathfrak{B}_\mathrm{R}) \oplus$
$(\mathfrak{A}_\mathrm{P} \wedge \mathfrak{A}_\mathrm{Q} \wedge \mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R})]$,

and its disjunctive normal form will be

$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R}) \vee$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \mathfrak{B}_\mathrm{R}) \vee$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R}) \vee$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \mathfrak{B}_\mathrm{Q} \wedge \mathfrak{B}_\mathrm{R}) \vee$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R}) \vee$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \mathfrak{B}_\mathrm{R}) \vee$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \mathfrak{B}_\mathrm{P} \wedge \mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R}) \vee$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R}) \vee$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \mathfrak{B}_\mathrm{R}) \vee$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R}) \vee$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R}) \vee$
$(\neg\mathfrak{A}_\mathrm{P} \wedge \mathfrak{A}_\mathrm{Q} \wedge \mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R}) \vee$
$(\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R}) \vee$
$(\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R}) \vee$
$(\mathfrak{A}_\mathrm{P} \wedge \neg\mathfrak{A}_\mathrm{Q} \wedge \mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R}) \vee$
$(\mathfrak{A}_\mathrm{P} \wedge \mathfrak{A}_\mathrm{Q} \wedge \neg\mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R}) \vee$
$(\mathfrak{A}_\mathrm{P} \wedge \mathfrak{A}_\mathrm{Q} \wedge \mathfrak{A}_\mathrm{R} \wedge \neg\mathfrak{B}_\mathrm{P} \wedge \neg\mathfrak{B}_\mathrm{Q} \wedge \neg\mathfrak{B}_\mathrm{R})$ = TRUE

□

□

Now, we are ready to propose the algorithm, solving the problem of optimal partitioning of a square with the cost function $f_c(T)$, formulated in the beginning of this section.

B. Selection of Optimal Parition

So far, we have identified three optimal partition shapes. For each of these shapes, there exist six possible mappings of three processors to the three parts of the shape, and these mappings have different costs. Therefore, for given problem parameters n, S_P, S_Q, and S_R, the straightforward algorithm would have to calculate costs of 6 mappings for each of the 3 partition shapes, resulting in a total of 18 potentially optimal partitions, and return the partition with the lowest cost.

Fortunately, we do not have to consider all possible mappings to find the optimal one. This follows from lemmas 1 – 3, which give us formulas for the cost of the optimal partition shapes. From formula (5), it is clear that for the Square Corner shape, any feasible mapping will have the fastest processor mapped to the P part. From formula (6), we can easily conclude that for the Square Rectangle shape any mapping, which maps the slowest processor to the R part, will be optimal. Similarly, for the Block Rectangle shape, formula (7) clearly indicates that any mapping, which maps the fastest processor to the P part, will be optimal. Therefore, instead of eighteen, our algorithm only calculates the cost of three parturitions, selecting the one with the lowest cost as the optimal.

VI. Experimental Validation

In this section, we present experiments validating our mathematical results. The main problem in the design of the experiments is how to accurately measure the contribution of data movements between the memories of the tightly coupled compute devices of our hybrid heterogeneous server in the total executing time. To the best of our knowledge, the experimental methodology proposed in the paper is the first that managed to solve this problem, at least, in the context of hybrid data-parallel applications. Solutions, which we found in literature, typically underestimate the contribution of the data movement and often give unstable results.

The main goal of the experiments is to validate the predictive accuracy of our theoretical model, thus demonstrating its usefulness in making practical decisions on partitioning square computational domains between tightly integrated compute devices in hybrid heterogeneous servers.

A. Experimental Methodology

We use a hybrid heterogeneous server, HCLServer01, in our experiments. HCLServer01 integrates an Intel Haswell multicore CPU, having 24 physical cores with 64 GB main memory, and two accelerators – one Nvidia K40c GPU and one Intel Xeon Phi 3120P. Detailed specifications of HCLServer01 are shown in Table.III.

HCLServer01 is the ideal platform for modelling a hybrid data-parallel application using three abstract interconnected heterogeneous processors. We abstract the GPU component of the application as abstract processor 1, the multi-core CPU component as processor 2 and the Xeon Phi component as processor 3. Each abstract processor thus represents a kernel of the data-parallel application running on the corresponding compute device and using both its and probably some CPU resources. The CPU abstract processor represents 22 cores of the multi-core CPU processor and DRAM involved in the execution of the CPU kernel. The GPU and PHI abstract processors each represents a CPU host-core and the accelerator together with its memory. We denote the GPU, CPU and PHI abstract processors by P, Q, R, which represents fast, medium, and slow processors respectively, and by S_P, S_Q, S_R – their relative speeds normalised to the square domain area. Assuming that the workload is optimally balanced between the processors according to their relative speeds, we focus exclusively on the communication cost in our experiments. We present three scenarios with different assumed relative speeds of the GPU, CPU and PHI abstract processors, resulting in a different $S_P : S_Q : S_R$ ratio for each case.

Intel Haswell E5-2670V3	
No. of cores per socket	12
Socket(s)	2
CPU MHz	1200.402
L1d cache, L1i cache	32 KB, 32 KB
L2 cache, L3 cache	256 KB, 30720 KB
Total main memory	64 GB DDR4
Memory bandwidth	68 GB/sec
NVIDIA K40c	
No. of processor cores	2880
Total board memory	12 GB GDDR5
L2 cache size	1536 KB
Memory bandwidth	288 GB/sec
Intel Xeon Phi 3120P	
No. of processor cores	57
Total main memory	6 GB GDDR5
Memory bandwidth	240 GB/sec

TABLE III: HCLServer1 specifications

We carefully design our experiments to accurately measure the time of the main-memory-to-main-memory data transfer between these heterogeneous processors, and consider this transfer time as communication time. This approach provides more realistic communication times as compared to approaches, which consider the communication time as the time of data transfer between virtual memories of the processes. We use a parallel matrix multiplication application, which computes the product of two dense square matrices. We comment out in this application the computation code to exclude the computation time from its execution time. For each scenario, matrices are partitioned between CPU, GPU and PHI in proportion to their assumed relative speeds and according to the three optimal partitioning shapes – Square Corner, Square Rectangle and Block Rectangle, as well as the commonly used 1D partitioning shape called Straight Line, and the execution time of the application for each partitioning shape is measured. The communication time of real experiments is then compared with the predicted communication time, calculated by using the accurate communication cost function and the average

communication bandwidth between the devices.

In our experimental platform, there are three communication links - between CPU and GPU, between CPU and PHI, and between PHI and GPU. Our model considers these links as homogeneous. In reality, the links between these processors are heterogeneous in nature, so we use the average bandwidth to calculate the model-predicted communication time.

We follow a statistical methodology to ensure reliability of our experimental results. For that, we make sure that the server is fully reserved and dedicated to these experiments only. We also monitor its load and check for any drastic fluctuation due to any abnormal event in the server. The application is repeatedly executed to obtain a data point, until the sample mean lies in the 95% confidence interval and a precision of 0.025 (2.5%) has been achieved. Student's t-test is used to determine the sample mean. The test assumes that the individual observations are independent and their population follows the normal distribution. We also allowed sufficient time to elapse between successive runs to make sure that cache effects and pipelining do not happen. We ensure that the problem size used for our experiments does not exceed the main memory of the compute devices and that paging does not occur.

B. Experimental Results

While our cost function assumes synchronous communications, we also experiment with asynchronous communications to demonstrate its predictive ability in this case. In all experiments, the problem size $N \times N$ is set to 22528×22528. The measured bandwidth of the CPU-GPU link is 9.7 GBps, the CPU-PHI link is 6.3 GBps, and the GPU-PHI link is 3.6 GBps. The calculated average bandwidth is 6.6 GBps.

In the first set of experiments, the speed ratio between GPU, CPU and PHI is assumed $1.0 : 0.5 : 2.5$. For this speed ratio, the model predicts Block Rectangle (BR) to be the optimal partition. Both synchronous and asynchronous communication experiments also validate that the Block Rectangle shape reports the lowest communication time as shown in Fig. 4.

For the second set of experiments, $S_P : S_Q : S_R$ is assumed $1.0 : 0.15 : 0.10$. The model predicts Square Corner (SC) to be optimal, and the real measurements also show that for both synchronous and asynchronous experiments, Square Corner is optimal Fig. 5. In the last set of experiments, $S_P : S_Q : S_R$ is assumed $1 : 0.7 : 0.10$. As shown in Fig. 6, both the model and the experiments identify Square Rectangle (SR) as optimal.

It is also evident from all the experiments that in the case of synchronous communications, the model predictions are sufficiently close to the real measurements in order to use them for accurate pairwise comparison of different partitions.

Our model predicts that Straight Line (SL) will never outperform any of the three optimal partitions. At the same time, its theoretical cost will be equal to the cost of Square Corner when $\sqrt{S_R} + \sqrt{S_Q} = n$, as it is in Scenario 1. In our experiments, we do see that the model predicted

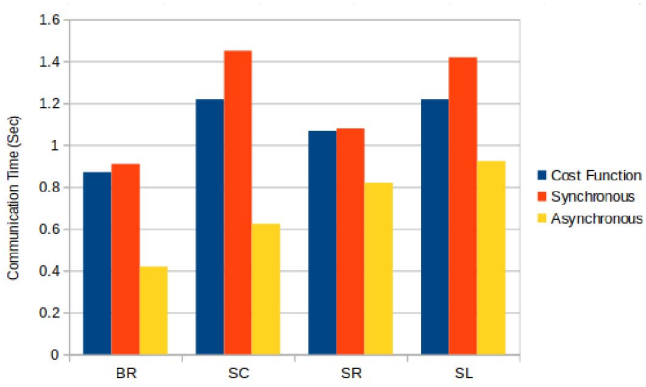


Fig. 4: Model-predicted and measured communication times of synchronous and asynchronous communication experiments for problem size 22528×22528 and speed ratio $S_P : S_Q : S_R$ = $1.0 : 0.5 : 0.25$.

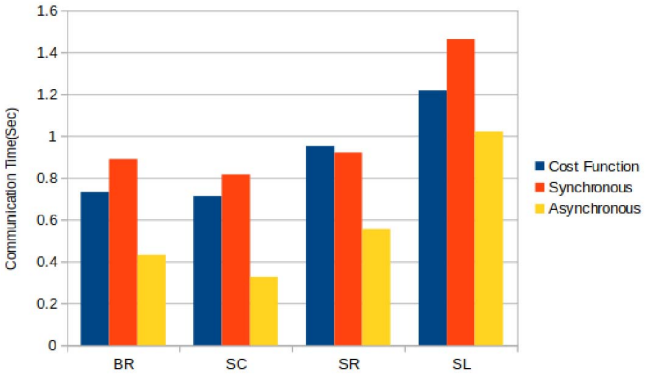


Fig. 5: Model-predicted and measured communication times of synchronous and asynchronous communication experiments for problem size 22528×22528 and speed ratio $S_P : S_Q : S_R$ = $1.0 : 0.15 : 0.10$.

communication time and the measured time of these two shapes in the case of synchronous communications are very close.

All together, the experimental results with synchronous communications demonstrate the accuracy of the proposed accurate communication cost function. The minor differences between the predicted communication times and the measured times are due to the use of the average bandwidth in the theoretical calculations. We believe that our model will accurately predict the time when the bandwidth of the communication links is the same. Unfortunately, we cannot validate it experimentally as in our experimental platform the communication links have different bandwidths because the execution of hybrid applications that use CPU, GPU and Xeon Phi will involve the CPU host-core, DRAM and PCIe's to transfer the data between CPU, GPU and Intel Xeon Phi. For example, data transfer between GPU and PHI passes through the CPU DRAM and PCIe's and will be slower than the data

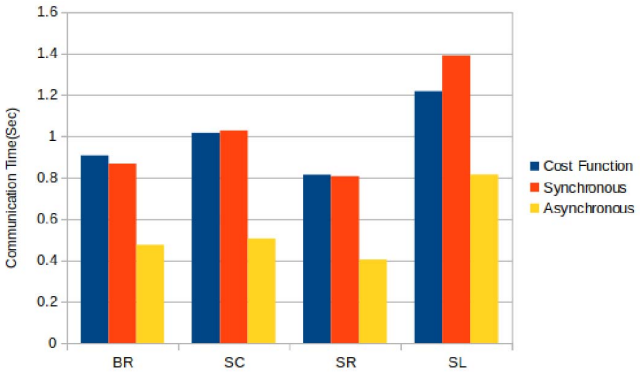


Fig. 6: Model-predicted and measured communication times of synchronous and asynchronous communication experiments for problem size 22528×22528 and speed ratio $S_P : S_Q : S_R$ = $1.0 : 0.7 : 0.10$.

transfer between GPU and CPU.

VII. Ongoing work and Future Direction

In this work, we solved the problem of optimal load - balanced partitioning of a square computation domain, which minimizes the total amount of data moved between three interconnected heterogeneous processors. The natural extension of this work is to solve the problem of optimal partitioning of a square computational domain between three heterogeneous processors interconnected by heterogeneous communication links. This is our ongoing research, and currently, we are working on this extended problem taking into account the bandwidth of communication links between the heterogeneous processors.

Acknowledgement

This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant Number 14/IA/2474.

References

[1] A. Kalinov and A. Lastovetsky, "Heterogeneous distribution of computations while solving linear algebra problems on networks of heterogeneous computers," *Lecture Notes in Computer Science*, vol. 1593, pp. 189–200, 04 1999.

[2] ——, "Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers," *Journal of Parallel and Distributed Computing*, vol. 61, no. 4, pp. 520–535, 2001.

[3] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix multiplication on heterogeneous platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1033–1051, 2001.

[4] R. Van De Geijn and J. Watts, "Summa: Scalable universal matrix multiplication algorithm," *Concurrency Computation*, vol. 9, no. 4, pp. 255–274, 4 1997.

[5] B. A. Becker and A. Lastovetsky, "Matrix multiplication on two interconnected processors," in *2006 IEEE International Conference on Cluster Computing*. IEEE, 2006, pp. 1–9.

[6] B. A. Becker and A. Lastovetsky, "Towards data partitioning for parallel computing on three interconnected clusters," in *Sixth International Symposium on Parallel and Distributed Computing (ISPDC'07)*, 2007, pp. 39–39.

[7] A. DeFlumere, A. Lastovetsky, and B. A. Becker, "Partitioning for parallel matrix-matrix multiplication with heterogeneous processors: The optimal solution," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, 2012, pp. 125–139.

[8] A. DeFlumere and A. Lastovetsky, "Searching for the optimal data partitioning shape for parallel matrix matrix multiplication on 3 heterogeneous processors," in *2014 IEEE International Parallel Distributed Processing Symposium Workshops*, May 2014, pp. 17–28.

[9] O. Beaumont, B. A. Becker, A. DeFlumere, L. Eyraud-Dubois, T. Lambert, and A. Lastovetsky, "Recent advances in matrix partitioning for parallel computing on heterogeneous platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 1, pp. 218–229, 2019.

[10] A. Lastovetsky and R. Reddy, "Data partitioning with a realistic performance model of networks of heterogeneous computers," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, IEEE Computer Society. Santa Fe, New Mexico, USA: IEEE Computer Society, 26-30 April 2004 2004, ¡p¿CD-ROM/Abstracts Proceedings¡/p¿.

[11] ——, "Data partitioning for multiprocessors with memory heterogeneity and memory constraints," *Scientific Programming*, vol. 13, no. 2, pp. 93–112, 2005.

[12] ——, "Data partitioning with a functional performance model of heterogeneous processors," *The International Journal of High Performance Computing Applications*, vol. 21, no. 1, pp. 76–90, 2007.

[13] ——, "Data distribution for dense factorization on computers with memory heterogeneity," *Parallel Computing*, vol. 33, no. 12, pp. 757–779, 2007.

[14] ——, "Two-dimensional matrix partitioning for parallel computing on heterogeneous processors based on their functional performance models," in *European Conference on Parallel Processing*. Springer, 2009, pp. 112–121.

[15] ——, "Distributed data partitioning for heterogeneous processors based on partial estimation of their functional performance models," in *European Conference on Parallel Processing*. Springer, 2009, pp. 91–101.

[16] A. Lastovetsky, L. Szustak, and R. Wyrzykowski, "Model-based optimization of eulag kernel on intel xeon phi through load imbalancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 787–797, 2016.

[17] A. Lastovetsky and R. Reddy Manumachu, "New model-based methods and algorithms for performance and energy optimization of data parallel applications on homogeneous multicore clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1119–1133, 2017.

[18] H. Khaleghzadeh, R. R. Manumachu, and A. Lastovetsky, "A novel data-partitioning algorithm for performance optimization of data-parallel applications on heterogeneous hpc platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 10, pp. 2176–2190, 2018.

[19] E. Dovolnov, A. Kalinov, and S. Klimov, "Natural block data decomposition for heterogeneous clusters," in *Proceedings International Parallel and Distributed Processing Symposium.* IEEE, 2003, pp. 10–pp.

[20] A. Lastovetsky, "On grid-based matrix partitioning for heterogeneous processors," in *Sixth International Symposium on Parallel and Distributed Computing (ISPDC'07).* IEEE, 2007, pp. 51–51.

[21] D. Clarke, A. Lastovetsky, and V. Rychkov, "Column-based matrix partitioning for parallel matrix multiplication on heterogeneous processors based on functional performance models," in *European Conference on Parallel Processing*. Springer, 2011, pp. 450–459.

[22] D. Clarke, A. Ilic, A. Lastovetsky, and L. Sousa, "Hierarchical partitioning algorithm for scientific computing on highly heterogeneous cpu+ gpu clusters," in *European Conference on Parallel Processing*. Springer, 2012, pp. 489–501.

[23] A. Fügenschuh, K. Junosza-Szaniawski, and Z. Lonc, "Exact and approximation algorithms for a soft rectangle packing problem," *Optimization*, vol. 63, no. 11, pp. 1637–1663, 2014.

[24] Z. Zhong, V. Rychkov, and A. Lastovetsky, "Data partitioning on multicore and multi-gpu platforms using functional performance models," *IEEE Transactions on Computers*, vol. 64, no. 9, pp. 2506–2518, 2014.

[25] A. DeFlumere and A. Lastovetsky, "Optimal data partitioning shape for matrix multiplication on three fully connected heterogeneous processors," in *European Conference on Parallel Processing*. Springer, 2014, pp. 201–214.

[26] O. Beaumont, L. Eyraud-Dubois, and T. Lambert, "A new approximation algorithm for matrix partitioning in presence of strongly heterogeneous

processors," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016, pp. 474–483.

[27] R. A. Van De Geijn and J. Watts, "Summa: Scalable universal matrix multiplication algorithm," *Concurrency: Practice and Experience*, vol. 9, no. 4, pp. 255–274, 1997.