Received: 3 March 2020 | Revised: 19 June 2020 | Accepted: 14 August 2020

DOI: 10.1002/cpe.6008

SPECIAL ISSUE PAPER

WILEY

About the granularity portability of block-based Krylov methods in heterogeneous computing environments

Luisa Carracciuolo[1] | Valeria Mele[2] | Lukasz Szustak[3]

[1]Institute of Polymers, Composites and Biomaterials (IPCB), Italian Research Council (CNR), Naples, Italy

[2]Department of Mathematics and Applications, University of Naples "Federico II", Naples, Italy

[3]Department of Computer Science, Czestochowa University of Technology, Czestochowa, Poland

Correspondence
Luisa Carracciuolo, c/o Comprensorio "Adriano Olivetti", Via Campi Flegrei 34 80078, Pozzuoli (NA), Italy.
Email: luisa.carracciuolo@cnr.it

Summary

Large-scale problems in engineering and science often require the solution of sparse linear algebra problems and the Krylov subspace iteration methods (KM) have led to a major change in how users deal with them. But, for these solvers to use extreme-scale hardware efficiently a lot of work was spent to redesign both the KM algorithms and their implementations to address challenges like extreme concurrency, complex memory hierarchies, costly data movement, and heterogeneous node architectures. All the redesign approaches bases the KM algorithm on block-based strategies which lead to the Block-KM (BKM) algorithm which has high granularity (i.e., the ratio of computation time to communication time). The work proposes novel parallel revisitation of the modules used in BKM which are based on the overlapping of communication and computation. Such revisitation is evaluated by a model of their granularity and verified on the basis of a case study related to a classical problem from numerical linear algebra.

KEYWORDS

iterative method, parallel computing, performance models, performance portability

1 | INTRODUCTION

Large-scale problems in engineering and science often require the solution of sparse linear algebra problems, such as systems of equations. The Krylov subspace iteration methods (KM) have led to a major change in how users deal with large, sparse, nonsymmetric matrix problems. Krylov methods can solve problems too large for other kinds of algorithms like factorizations, as well as problems where the coefficient matrix $\mathcal{A}$ is only available as a function performing the matrix by vector multiplication operation. For all these reasons, the Krylov methods can be listed among the top 10 algorithms for Computing in Science and Engineering.[1] On these methods, computational scientists continue in investing also to make them usable on future computing systems (i.e., see the PEEKS initiative[2] of the Exascale Computing project[3]).

However, the performance of the Krylov methods is often dominated by communication, as communication has become much more expensive compared with computation, in terms of both throughput and energy consumption. As in Yamazaki et al.,[4] the term *communication* can be used to include both *horizontal* data movement between parallel processing units, as well as *vertical* data movement between memory hierarchy levels. In fact, in their original formulation, these methods are based on level 1 BLAS operations[5] (i.e., vector products, products of a scalar by a vector, etc.). Such operations have a low granularity and they fail to guarantee good performance especially in high-performance computing contexts. In parallel computing, the term granularity of a task is a measure of the amount of work (or computation) which is performed by that task.[6] Such a measure intends to take into account the ratio of computation time to communication time.

For these solvers to use extreme-scale hardware efficiently, during the last three decades, a lot of work was spent to redesign both the KM algorithms and their implementations (e.g., see Yamazaki et al.,[4] Bai et al.,[7] Hoemmen,[8] Ghysels et al.,[9] Mohiyuddin et al.,[10] and Imberti et al.[11]) to try to address challenges like extreme concurrency, complex memory hierarchies, costly data movement, and heterogeneous node architectures.

All the redesign approaches base the algorithms on BLAS 2 and 3 operations (products of a vector by a matrix, matrices products, etc.) which have a higher granularity.

The BLAS-based algorithm formulation allows not only the reuse of procedures from optimized software libraries for different kinds of computing architectures (e.g., see the MAGMA software library[12] for GPU based computing systems, or the ScaLAPACK software library[13] for distributed memory systems) but also the *portability* of algorithm implementation. In the last years, the *portability* term has enriched itself with new meanings: computational and computing scientists are questioning and confronting each other about how to measure the degree to which an application (or library, programming model, algorithm implementation, etc.) has become *performance portable*. The terms *performance portability* has been informally used in computing communities to substantially refer to (1) the ability to run one application across multiple hardware platforms and (2) achieving some notional level of performance on these platforms.[14] Among the efforts related to the *performance portability* issue should be certainly cited in the annual performance portability workshops organized by the US Department of Energy.[15] We strongly believe that this will be one of the hottest points in future, for many interesting application fields, like Machine Learning of course, but also the High-Performance Computing, Cloud Computing and the Internet of Things, where architectures are complex, highly heterogeneous and overall continuing evolving (as you can read in Laccetti et al.[16]), or the Parallelims-in-Time that today can benefit from the new high-performance architectures and libraries, as the authors discussed in Carracciuolo et al.[17] and Mele et al.[18,19]

This work intends to deal with the issue related to the evaluation of performance portability of KM block-based algorithms on the computing systems which will respond to the new requirements of the incoming exascale era (e.g., see The Exascale Computing Project (ECP)[3] or the European Horizon 2020 *FET Proactive—High-Performance Computing* Call[20]). Most likely, these systems will respond to the following description: multinode systems where each node will have a high level of internal parallelism which will be also made available by technologies such as NVIDIA GPU and Intel Xeon Phi. In particular, as in Carracciuolo and Lapegna[21] this work intends to analyze the portability of some performance metrics of a *parallel* implementation of a KM block-based algorithm on heterogeneous CPU–GPU systems equipped with standard scientific libraries as MAGMA.[12] The considered implementation proposes novel revisitation of well-known algorithms, such as those used to compute the QR factorization, which is based on the overlapping of communication and computation.

The work is organized as follows: in Section 2 the block-based version of KM method (BKM) is presented; in Section 3 some new parallel implementation of BKM are described and in Section 4 a model able to describe the implementation performance in terms of its granularity is given; in Section 5 the model is verified on the basis of a case study related to a classical problem from numerical linear algebra; in Section 6 results are summarized and some future work is described.

2 | THE KM BLOCK-BASED ALGORITHMS

Let $\mathcal{A}$ be an $n \times n$ real matrix and, let us recall from Saad [22] that KM builds, by an iterative approach, an approximation x_m of the solution of the linear system

$$\mathcal{A}x = b, \tag{1}$$

that is extracted from a subspace $x_0 + \mathcal{K}_m$ of $\mathbb{R}^n$ under the condition that $b - \mathcal{A}x_m \perp \mathcal{L}_m$ where:

1. x_0 represents an arbitrary initial guess to the solution,
2. $\mathcal{K}_m = \mathcal{K}_m(\mathcal{A}, r_0) = span\{r_0, \mathcal{A}r_0, \ldots, \mathcal{A}^{m-2}r_0, \mathcal{A}^{m-1}r_0\}$ represents the subspace of $\mathbb{R}^n$ spanned by a suitable set of monomials $M_{j=0,\ldots,m-1}(\mathcal{A})$ of the matrix $\mathcal{A}$ and
3. $r_0 = b - \mathcal{A}x_0$ represents the initial residual.

If $\mathcal{L}_m = \mathcal{A}\mathcal{K}_m$, the iterative algorithm that computes the new approximation x_{m+1} at the step $m+1$ from the previous one $m \geq 0$ was known as *Generalized Minimum Residual Method (GMRES)*[22] and it performs the actions listed in Algorithm 1 where $\beta = \|r_0\|_2, v_0 = q_0 = r_0/\beta, e_1 = [1, 0, 0, \ldots\ 0, 0]^T, Q_0 = [q_0]$, columns of Q_m are mutually orthonormal and $\underline{H}_{m+1}$ is a Hessenberg matrix.

A block-based version of Algorithm 1 is available and, during the last decades, it was referenced using different names: from *s-step GMRES*[23] to a more recent *Communication Avoiding-CA GMRES*.[8]

The block-based version of Algorithm 1 is listed in Algorithm 2 where:

$$v_{m+j} = \prod_{i=0}^{j} M_i(\mathcal{A})v_m, \quad j = 1, \ldots, s \tag{2}$$

$$\mathfrak{Q}_0 = [v_0, \ldots, v_{s-1}, v_s], \tag{3}$$

Algorithm 1. The mth step of GMRES method

1: **OrthoBegin** ▷ Build an orthonormal basis of column vectors $Q_{m+1} = [Q_m, q_m]$ of $\mathcal{K}_{m+1}$
2: Compute $v_{m+1} = M_m(\mathcal{A})v_m$
3: Orthonomalize v_{m+1} againstQ_m to compute q_m and$\underline{H}_{m+1}$ such that $\mathcal{A}Q_m = Q_{m+1}\underline{H}_{m+1}$
4: **OrthoEnd**
5: **SolBegin** ▷ Extract a suitable vector from a subspace $x_0 + \mathcal{K}_{m+1}$
6: compute $y_{m+1} = \text{argmin}_y \left\| \beta e_1 - \underline{H}_{m+1} y \right\|_2$
7: compute $x_{m+1} = x_0 + Q_{m+1} y_{m+1}$
8: **SolEnd**
9: **PrepBegin** ▷ Prepare for the next step
10: Assign $v_{m+1} \leftarrow q_m$
11: **PrepEnd**

$$\mathfrak{Q}_{m+1} = [q_0, \ \dots \ , q_{ms-1}, q_{ms}, q_{ms+1}], \tag{4}$$

$$\overline{\mathfrak{Q}}_{m+1} = [q_0, \ \dots \ , q_{ms-1}, q_{ms}], \tag{5}$$

$$V_{m+1} \in \mathbb{R}^{n \times s}, \tag{6}$$

$$\mathfrak{Q}_{m+1} \in \mathbb{R}^{n \times (ms+1)}, \tag{7}$$

$$Q_{m+1} \in \mathbb{R}^{n \times s}, \tag{8}$$

$$R_{m+1} \in \mathbb{R}^{s \times s}, \tag{9}$$

$$\mathfrak{H}_{m+1} \in \mathbb{R}^{(ms+1) \times ms}. \tag{10}$$

This version of GMRES is essentially based on a block version of operations described in lines 2 and 3 of Algorithm 1. In the new version (see line 2 of Algorithm 2) multiple column vectors $V_{m+1} = [v_{m+1}, \ \dots \ , v_{m+s}]$ are calculated at the same step m and the new orthonormal basis is computed by two normalizations steps:

1. orthonormalize V_{m+1} against the orthonormal basis $\mathfrak{Q}_m$ and computes $\overline{V}_{m+1}$,
2. orthonormalize columns of $\overline{V}_{m+1}$ by a QR factorization.[24]

More details on Algorithm 2 can be found in Hoemmen.[8]

Algorithm 2. The mth step of block-based GMRES method

1: **OrthoBegin** ▷ Build an orthonormal basis of column vectors $\mathfrak{Q}_{m+1} = [\mathfrak{Q}_m, Q_{m+1}]$ of $\mathcal{K}_{m+s}$
2: Compute $V_{m+1} = [v_{m+1}, \dots , v_{m+s-1}, v_{m+s}]$
3: Compute $\mathfrak{R}_{m+1} = \mathfrak{Q}_m^T V_{m+1}$
4: Compute $\bar{V}_{m+1} = V_{m+1} - \mathfrak{Q}_m \mathfrak{R}_{m+1}$
5: Compute the QR factorization of $\bar{V}_{m+1} = Q_{m+1} R_{m+1}$
6: Compute the Hessenberg matrix $\underline{\mathfrak{H}}_{m+1}$ (from R_{m+1} and $\mathfrak{R}_{m+1}$) such that$\mathcal{A}\bar{\mathfrak{Q}}_{m+1} = \mathfrak{Q}_{m+1}\underline{\mathfrak{H}}_{m+1}$
7: **OrthoEnd**
8: **SolBegin** ▷ Extract a suitable vector from a subspace $x_0 + \mathcal{K}_{m+s}$
9: compute $y_{m+1} = \text{argmin}_y \left\| \beta e_1 - \underline{\mathfrak{H}}_{m+1} y \right\|_2$
10: compute $x_{m+1} = x_0 + \mathfrak{Q}_{m+1} y_{m+1}$
11: **SolEnd**
12: **PrepBegin** ▷ Prepare for the next step
13: Assign $v_{m+1} \leftarrow q_{ms+1}$
14: **PrepEnd**

In the exact arithmetic, s steps of the Algorithm 1 are equivalent to one step of the Algorithm 2 but the Algorithm 2 suffers from some instabilities when finite precision is used. A lot of work was spent during the last years to identify the origin of these phenomena and to mitigate their

consequences (e.g., see Bai et al.[7] about the role of monomials $M_j(\mathcal{A})$ on the "condition number" of V_{m+1} and then on the convergence of Algorithm 2. Also to temper instability effects, s is chosen to be $s \ll n$ and the execution of Algorithm 2 is restarted[22] just after some steps $m \ll n$. By under these last assumptions and from (2) to (10) follows that the operations of Algorithm 2 with higher computational costs are those listed at lines 2–5 and 10.

In the next Section 3 some strategies for the distribution, on different P tasks, of the computational load of the above listed operations are analyzed and a parallel version of Algorithm 2 is described.

3 | THE PARALLEL VERSION OF BLOCK-BASED KM METHOD

In this section, we describe some parallelization strategies for operations with higher computational costs in Algorithm 2 and finally propose the *parallel version* of the block-based KM method.

3.1 | V_{m+1} orthonormalization against $\mathfrak{Q}_m$

The operations at lines 3 and 4 of the Algorithm 2 can be rewritten respectively as:

$$\begin{aligned}\mathfrak{R}_{m+1} = \mathfrak{Q}_m^T V_{m+1} &= \begin{pmatrix} {\mathfrak{Q}_m^1}^T & {\mathfrak{Q}_m^2}^T & \dots & {\mathfrak{Q}_m^{P-1}}^T & {\mathfrak{Q}_m^P}^T \end{pmatrix} \begin{pmatrix} V_{m+1}^1 \\ V_{m+1}^2 \\ \vdots \\ V_{m+1}^{P-1} \\ V_{m+1}^P \end{pmatrix} \\ &= \sum_{p=1}^{P} {\mathfrak{Q}_m^p}^T V_{m+1}^p \end{aligned} \tag{11}$$

and

$$\begin{aligned}\begin{pmatrix} \overline{V}_{m+1}^1 \\ \overline{V}_{m+1}^2 \\ \vdots \\ \overline{V}_{m+1}^{P-1} \\ \overline{V}_{m+1}^P \end{pmatrix} &= \overline{V}_{m+1} = V_{m+1} - \mathfrak{Q}_m \mathfrak{R}_{m+1} \\ &= \begin{pmatrix} V_{m+1}^1 \\ V_{m+1}^2 \\ \vdots \\ V_{m+1}^{P-1} \\ V_{m+1}^P \end{pmatrix} - \begin{pmatrix} \mathfrak{Q}_m^1 \\ \mathfrak{Q}_m^2 \\ \vdots \\ \mathfrak{Q}_m^{P-1} \\ \mathfrak{Q}_m^P \end{pmatrix} \mathfrak{R}_{m+1} \\ &= \begin{pmatrix} V_{m+1}^1 \\ V_{m+1}^2 \\ \vdots \\ V_{m+1}^{P-1} \\ V_{m+1}^P \end{pmatrix} - \begin{pmatrix} \mathfrak{Q}_m^1 \mathfrak{R}_{m+1} \\ \mathfrak{Q}_m^2 \mathfrak{R}_{m+1} \\ \vdots \\ \mathfrak{Q}_m^{P-1} \mathfrak{R}_{m+1} \\ \mathfrak{Q}_m^P \mathfrak{R}_{m+1} \end{pmatrix} \\ &= \begin{pmatrix} V_{m+1}^1 \\ V_{m+1}^2 \\ \vdots \\ V_{m+1}^{P-1} \\ V_{m+1}^P \end{pmatrix} - \begin{pmatrix} \mathfrak{Q}_m^1 \mathfrak{R}_{m+1}^1 \\ \mathfrak{Q}_m^2 \mathfrak{R}_{m+1}^2 \\ \vdots \\ \mathfrak{Q}_m^{P-1} \mathfrak{R}_{m+1}^{P-1} \\ \mathfrak{Q}_m^P \mathfrak{R}_{m+1}^P \end{pmatrix} - \begin{pmatrix} p1 \neq \sum \mathfrak{Q}_m^1 \mathfrak{R}_{m+1}^1 \\ p2 \neq \sum \mathfrak{Q}_m^2 \mathfrak{R}_{m+1}^2 \\ \vdots \\ pP1 - \sum \mathfrak{Q}_m^{P-1} \mathfrak{R}_{m+1}^{P-1} \\ pP \neq \sum \mathfrak{Q}_m^P \mathfrak{R}_{m+1}^P \end{pmatrix}, \end{aligned} \tag{12}$$

where $\mathfrak{R}^p_{m+1}$ stands for $\mathfrak{Q}^{p\,T}_m V^p_{m+1}$ for all $p = 1, \ldots, P$.

Using (11) and (12), a distribution on P tasks of the computational load of related operations can be performed, on each p of P tasks, using the following Algorithm 3 where computational phases are overlapped with communication ones.

Algorithm 3. A parallel algorithm for the first step of the orthonormalization phase during the mth step of block-based KM

1: Compute $\mathfrak{R}^p_{m+1} = \mathfrak{Q}^{p\,T}_m V^p_{m+1}$ ▷ p is the identification number of the executing task among the P total tasks
2: Assign $\mathfrak{R}_{m+1} \leftarrow \mathfrak{R}^p_{m+1}$
3: Execute a Non-Blocking Allgather of $\mathfrak{R}^p_{m+1}$
4: Compute $\bar{V}^p_{m+1} = V^p_{m+1} - \mathfrak{Q}^p_m \mathfrak{R}^p_{m+1}$
5: **for all** $j \neq p$ **do**
6: **if** ($\mathfrak{R}^j_{m+1}$ is available **then**)
7: Compute $\bar{V}^p_{m+1} = \bar{V}^p_{m+1} - \mathfrak{Q}^p_m \mathfrak{R}^j_{m+1}$
8: Compute $\mathfrak{R}_{m+1} = \mathfrak{R}_{m+1} + \mathfrak{R}^j_{m+1}$
9: **end if**
10: **end for**

3.2 | Computation of QR factorization of $\overline{V}_{m+1}$

In this section, we show how *Tall Skinny QR (TSQR)*, a communication-avoiding QR factorization for dense matrices with many more rows than columns,[8] can be used to parallelize the operation at line 5 of the Algorithm 2. TSQR, using a block approach on a binary tree, can compute the QR factorization of the $n \times s$ matrix A, with $n \gg s$ in $l = \log_2 P$ stages when P is the number of subblocks $(A_i)_i = 1, \ldots, P$ of A and where each subblock is a $n_i \times s$ matrix (we can assume that $n_i \approx n/P$).

In Figure 1 is represented an example of a TSQR execution when $P = 8$ where the gray boxes indicate where local QR factorizations take place. The Q and R factors each have both apex and subscript (the apex is the stage number $j = 0, \ldots, l$, the subscript is the sequence number $i = 1, \ldots, 2^{l-j}$ for that stage) and:

1. $Q^0_i R^0_i$ is the QR factorization of the ith block A_i of A, $\forall i = 1, \ldots, P$;
2. $Q^j_i R^j_i$ is the QR factorization of the ith block $\begin{pmatrix} R^{j-1}_{2*(i-1)+1} \\ R^{j-1}_{2*(i-1)+2} \end{pmatrix}$, $\forall i = 1, \ldots, 2^{l-j}, \forall j = 1, \ldots, l$;
3. Q^0_i is a $n_i \times s$ matrix, $\forall i = 1, \ldots, P$;
4. Q^j_i is a $2s \times s$ matrix, $\forall i = 1, \ldots, 2^{l-j}, \forall j = 1, \ldots, l$;
5. R^j_i is a $s \times s$ matrix, $\forall i = 1, \ldots, 2^{l-j}, \forall j = 0, \ldots, l$.

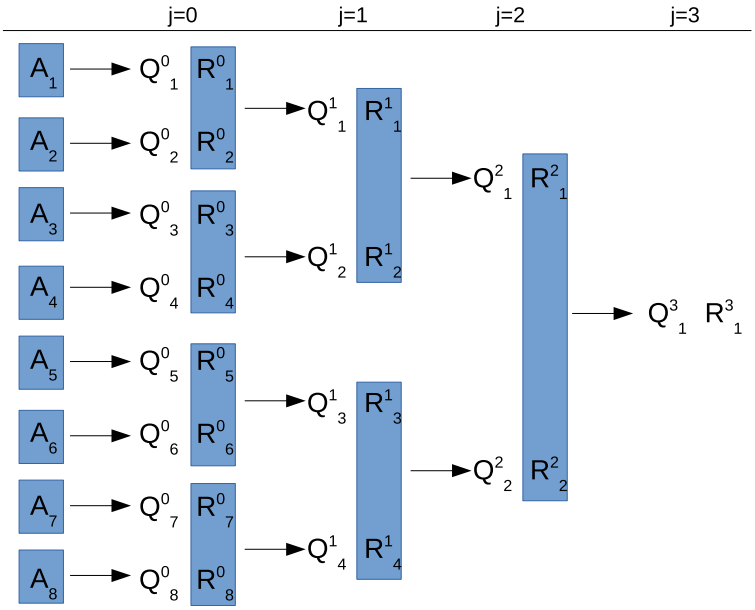

FIGURE 1 Representation of TSQR execution on $P = 8$ tasks

Then it follows that:

$$A = \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_{P-1} \\ A_P \end{pmatrix} = \left(\prod_{j=0}^{l} diag\left(Q^j_{i=1,\dots,2^{l-j}} \right) \right) R^l_1, \tag{13}$$

where $diag\left(Q^j_{i=1,\dots,2^{l-j}} \right)$ is the block diagonal matrix

$$diag(Q^j_{i=1,\dots,2^{l-j}}) = \begin{pmatrix} Q^j_1 & 0 & \dots & 0 & 0 \\ 0 & Q^j_2 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & Q^j_{2^{l-j}-1} & 0 \\ 0 & 0 & \dots & 0 & Q^j_{2^{l-j}} \end{pmatrix},$$

whose dimensions are: $n \times Ps$, if $j = 0$, and $2^{l-j+1}s \times 2^{l-j}s$, if $j = 1, \dots, l$.

For all of the above, in order to define an algorithm to compute the QR factorization of

$$A = \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_{P-1} \\ A_P \end{pmatrix},$$

from (13) we obtain that:

$$\begin{pmatrix} Q_1 \\ Q_2 \\ \vdots \\ Q_{P-1} \\ Q_P \end{pmatrix} = Q = Q^0 Q^l = \begin{pmatrix} Q^0_1 Q^l \\ Q^0_2 Q^l \\ \vdots \\ Q^0_{P-1} Q^l \\ Q^0_P Q^l \end{pmatrix}$$

$$R = R^l_1,$$

where

$$Q^l = \prod_{j=1}^{l} diag\left(Q^j_{i=1,\dots,2^{l-j}} \right)$$

is a $Ps \times s$ matrix and where $Q^0 = diag\left(Q^0_{i=1,\dots,P} \right)$ is a $n \times Ps$ matrix.

We then observe that the main computational cost of TSQR algorithm is related to the definition of matrix Q^0 since matrix Q^l has very small dimensions compared with those of Q^0. A parallel algorithm, that is going to be executed on each p of P tasks and that takes into account this last observation, is the one listed in Algorithm 4.

Algorithm 4. A parallel algorithm for TSQR computation of A

1: Compute QR factorization $Q_p^0 R_p^0$ of A_p ▷ p is the identification number of the executing task among the P total tasks
2: **if** $(p = 1)$ **then**
3: Gather $R_j^0, \forall j$
4: Compute Q^I matrix
5: Broadcast Q^I matrix to all P tasks
6: **end if**
7: Compute $Q_p = Q_p^0 Q^I$

3.3 | Computation of the columns of V_{m+1}

In this section, we show a parallel algorithm that computes the matrix-vector products

$$^{(s)}q = \mathcal{A}^s b, \quad s = 1, \dots, S \tag{14}$$

on which the computation of the S columns of V_{m+1} is based.

Such algorithm (see Algorithm 5) computes in each step s of a cycle the two successive vectors $^{(s)}q$ and $^{(s+1)}q$ by a flow of operations that overlap computation and communication phases. The matrix $\mathcal{A}$ and vectors b and $^{(s)}q$, $s = 1, \dots, S$ are distributed, in a block-row fashion, among P tasks, that is, if

$$\mathcal{A} = \begin{pmatrix} \mathcal{A}_{1,1} & \dots & \mathcal{A}_{1,P} \\ \vdots & \ddots & \vdots \\ \mathcal{A}_{P,1} & \dots & \mathcal{A}_{P,P} \end{pmatrix}, \tag{15}$$

$$b = \begin{pmatrix} b_1 \\ \vdots \\ b_P \end{pmatrix}, \tag{16}$$

$$^{(s)}q = \begin{pmatrix} ^{(s)}q_1 \\ \vdots \\ ^{(s)}q_P \end{pmatrix}, \tag{17}$$

then the subblocks $\{\mathcal{A}_{p,j}\}_{j=1,\dots,P}$, b_p and $^{(s)}q_p$, $s = 1, \dots, S$ are assigned to the pth task.

To efficiently compute (14), in a parallel context when $\mathcal{A}$ is a very sparse matrix, the issue of a good distribution of the matrix elements among the parallel tasks have to be considered. Such distribution should be able to guarantee both a high computational load per task and a low communication overhead. A way to reach the goal of getting the right matrix elements distribution is based on techniques using hypergraph partition models. Let us consider the *column-net hypergraph model*[25] $\mathcal{H}_{\mathcal{R}} = (\mathcal{V}_{\mathcal{R}}, \mathcal{N}_{\mathcal{C}})$ of the matrix $\mathcal{A}$: a vertex $v_i \in \mathcal{V}_{\mathcal{R}}$ and a net $n_j \in \mathcal{N}_{\mathcal{C}}$ exist for each row r_i and column c_j $\mathcal{A}$, respectively. A net $n_j \subseteq \mathcal{V}_{\mathcal{R}}$ contains the vertices corresponding to the rows that have a nonzero entry in column c_j. A weight w_i is assigned to the vertex $v_i \in \mathcal{V}_{\mathcal{R}}$ and it is set to the total number of nonzeros in row r_i. Given the hypergraph $\mathcal{H}_{\mathcal{R}} = (\mathcal{V}_{\mathcal{R}}, \mathcal{N}_{\mathcal{C}}), \Pi = \{V_R^1, \dots, V_R^K\}$ is called a *K-way partition* of the vertex set $\mathcal{V}_{\mathcal{R}}$ if

$$V_R^k \neq \emptyset, \quad k = 1, \dots, K, \tag{18}$$

$$V_R^k \cap V_R^l = \emptyset, \quad k, l = 1, \dots, K, \ k \neq j, \tag{19}$$

$$\cup_k V_R^k = \mathcal{V}_{\mathcal{R}}. \tag{20}$$

A *K-way partition* vertex Π of $\mathcal{H}_{\mathcal{R}}$ is said to satisfy the partitioning constraint if

$$W_k \leq W_{avg}(1 + \varepsilon), \quad k = 1, \dots, K, \tag{21}$$

where $W_k = \sum_{v_i \in V_R^k} w_i$, $W_{avg} = \frac{\sum_k W_k}{K}$ and ε is the allowable imbalance ratio.

Algorithm 5. A parallel algorithm for the computation of the S matrix-vector products ${}^{(s)}q = \mathcal{A}^s b, s = 1, \ldots, S$

```
1: Assign s ← 1                                   ▷ p is the identification number of the executing task among the P total tasks
2: while s ≤ S do
3:     Assign y_p^nloc ← 0
4:     Assign x_p ←^s b
5:     Execute a Non-Blocking Allgather of x_p
6:     Compute y_p^loc = A_{p,p} x_p
7:     if (s + 1 ≤ S then)
8:         Compute z_p = A_{p,p} y_p^loc
9:     end if
10:    for all j ≠ p do
11:        if (x_j is available then)
12:            Compute y_p^nloc = y_p^nloc + A_{p,j} x_j
13:        end if
14:    end for
15:    Compute y_p = y_p^loc + y_p^nloc
16:    Assign ^(s)q_p ← y_p
17:    if (s + 1 ≤ S then)
18:        Execute a Non-Blocking Allgather of y_p
19:        Compute z_p = z_p + A_{p,p} y_p^nloc
20:        for all j ≠ p do
21:            if (y_j is available then)
22:                Compute z_p = z_p + A_{p,j} y_j
23:            end if
24:        end for
25:        Assign ^(s+1)q_p ← z_p
26:    end if
27:    s ← s + 2
28: end while
```

Let us denote by $\mathcal{N}_{\mathcal{E}}$ the set of external nets of a partition Π where a net n_j is said to be external if it connects more than one part of Π (i.e., one of its vertexes is a member of more than one part V_R^k).

This model could be used to find a "good" row block distribution of matrix $\mathcal{A}$ by the solution of the following constrained optimization problem:

Problem 1. Compute the *partitioning objective* Π_{opt} as

$$\Pi_{opt} = \text{argmin}_{\Pi}\text{EdgeCutSize}(\Pi), \tag{22}$$

where EdgeCutSize(Π) is the following function defined over the set of all the *K-way partition* vertex Π of $\mathcal{H}_{\mathcal{R}}$ satisfying the partitioning constraint

$$\text{EdgeCutSize}(\Pi) = n_j \mathcal{N}_{\mathcal{E}} \in \sum(l_j - 1) \tag{23}$$

and where l_j is the *connectivity* of net n_j (i.e., the number of parts connected by n_j).

To any *partioning* action could be associated with a matrix row and column reordering, for example, let us suppose that $I_p = \{{}^p i_1, \ldots, {}^p i_{n_p}\}$ is the set of row indices of matrix K_λ assigned to the pth partition then the permutation π represented in two-line form by

$$\pi = \begin{pmatrix} 1 & \ldots & n_1 & \ldots & \sum_{i=1}^{P-1} n_i & \ldots & n \\ {}^1 i_1 & \ldots & {}^1 i_{n_1} & \ldots & {}^P i_1 & \ldots & {}^P i_{n_P} \end{pmatrix}$$

define the permutation matrix P_π,[24] where $n_p = |I_p|$. With the terms "partition-based reordered matrix" ${}^\pi\mathcal{A}$ we meant the matrix whose rows and columns are permuted by the P_π permutation matrix, that is, ${}^\pi\mathcal{A} = P_\pi \mathcal{A} P_\pi$. The block distribution of consecutive rows of the *partition-based reordered matrix* ${}^\pi\mathcal{A}$ satisfies the properties to be well balanced (see the partitioning constraint 21) and with a low communication overhead (see the partitioning objective 22).

3.4 | The whole parallel method

Suppose that x_m, and then columns of V_m, $\overline{V}_m$, and $\mathfrak{Q}_m$, are distributed in a block-row fashion on different P tasks $p = 1, \dots, P$ and that, for each m, x_m^p, V_m^p, $\overline{V}_m^p$, and $\mathfrak{Q}_m^p$ denote the pth block of x_m, V_m, $\overline{V}_m$, and $\mathfrak{Q}_m$, respectively. The above-mentioned data are all related to the operations with the higher computational cost which is the sole requiring a computational load distribution. Also, suppose that to an additional task $p = 0$ are delegated all the operations less computationally expensive. Then the final parallel version of the Algorithm 2 could be as listed in Algorithm 6 executable on each $p = 0, \dots, P$ of $P + 1$ tasks. We also note that the instructions listed at lines 31–39, as described in Reference 8, could be executed only if iterative block method converged.

Algorithm 6. The parallel mth step of block-based GMRES method

1: **OrthoBegin**
2: **if** $(p > 0)$ **then**
3: Compute $V_{m+1}^p = \left[v_{m+1}^p, \dots, v_{m+s-1}^p, v_{m+s}^p\right]$ by the Algorithm 5
4: **end if**
5: **if** $(p > 0)$ **then**
6: Compute $\mathfrak{R}_{m+1}^p = \mathfrak{Q}_m^{p\,T} V_{m+1}^p$ andAssign $\mathfrak{R}_{m+1} \leftarrow \mathfrak{R}_{m+1}^p$
7: Execute a Non-Blocking Allgather of $\mathfrak{R}_{m+1}^p$
8: Compute $\bar{V}_{m+1}^p = V_{m+1}^p - \mathfrak{Q}_m^p \mathfrak{R}_{m+1}^p$
9: **for all** $j \neq p$ **do**
10: **if** ($\mathfrak{R}_{m+1}^j$ is available **then**)
11: Compute $\bar{V}_{m+1}^p = \bar{V}_{m+1}^p - \mathfrak{Q}_m^p \mathfrak{R}_{m+1}^j$
12: Compute $\mathfrak{R}_{m+1} = \mathfrak{R}_{m+1} + \mathfrak{R}_{m+1}^j$
13: **end if**
14: **end for**
15: **else**
16: Get $\mathfrak{R}_{m+1}$ from task $p = 1$
17: **end if**
18: **if** $(p > 0)$ **then**
19: Compute QR factorization $Q_p^0 R_p^0$ of $\bar{V}_{m+1}^p$
20: **else**
21: Gather $R_j^0, \forall j$
22: Compute Q^I matrix and Broadcast to all tasks $p = 1, \dots, P$
23: $R_{m+1} \leftarrow R_1^I$
24: **end if**
25: **if** $(p > 0)$ **then**
26: Compute $Q_p = Q_p^0 Q^I$
27: **else**
28: Compute the Hessenberg matrix $\underline{\mathfrak{H}}_{m+1}$ from R_{m+1} and $\mathfrak{R}_{m+1}$
29: **end if**
30: **OrthoEnd**
31: **SolBegin**
32: **if** $(p = 0)$ **then**
33: compute $y_{m+1} = \text{argmin}_y \left\| \beta e_1 - \underline{\mathfrak{H}}_{m+1} y \right\|_2$
34: Broadcast y_{m+1} to all tasks $p = 1, \dots, P$
35: **end if**
36: **if** $(p > 0)$ **then**
37: compute $x_{m+1}^p = x_0^p + \mathfrak{Q}_{m+1}^p y_{m+1}$
38: **end if**
39: **SolEnd**
40: **PrepBegin**
41: **if** $(p > 0)$ **then**
42: Assign $v_{m+1}^p \leftarrow q_{ms+1}^p$
43: **end if**
44: **PrepEnd**

4 | MODELS FOR PERFORMANCE ANALYSIS OF A PARALLEL BLOCK-BASED KM METHOD

This work intends to deal with the issue related to the evaluation of the *performance portability* of KM block-based algorithms. To do this, we want to measure the performance of the proposed algorithm in terms of its granularity ${}^{(Sys)}G(m, s, n, \varphi)$, defined as

$$ {}^{(Sys)}G(m, s, n, \varphi) = \frac{{}^{(Sys)}T_{comp}(m, s, n, \varphi)}{{}^{(Sys)}T_{comm}(m, s, n, \varphi)}, \tag{24} $$

where ${}^{(Sys)}T_{comp}(m, s, n, \varphi)$ and ${}^{(Sys)}T_{comm}(m, s, n, \varphi)$ represent the time spent in computation and communication, respectively.

We already noticed in the first section that the performance of the Krylov methods is often dominated by communication, in terms of both throughput and energy consumption. When these methods are based on level 1 BLAS operations,[5] which have a low granularity, they fail to guarantee good performance especially in high-performance computing contexts. Thus all the new redesign approaches base the algorithms on BLAS 2 and 3 operations (products of a vector by a matrix, matrices products, etc.) which have a higher granularity.

If we consider just the part of the algorithm that omits the instructions performed only on condition (i.e., lines range 31-39), ${}^{(Sys)}T_{comp}$ and ${}^{(Sys)}T_{comm}$ respectively should be:

$$ \begin{aligned} {}^{(Sys)}T_{comp}(m, s, n, \varphi) \approx & \underbrace{{}^{(Sys)}\tau_{comp}^{p>0}\left[P\left(ms^2\frac{n}{P} + s\frac{n}{P}\right) + ms^2\frac{n}{P} + (P-1)ms^2\right]}_{\text{ortho.ph.: step 1}} \\ & + \underbrace{{}^{(Sys)}\tau_{comp}^{p>0}\left[(P+2)s^2\frac{n}{P}\right] + \tau_{comp}^{p=0}[O(Ps^3) + O(m^3s^3)]}_{\text{ortho.ph.: step 2}} + \underbrace{{}^{(Sys)}\tau_{comp}^{p>0}\left[s\varphi n\frac{n}{P} + s(P-1)\frac{n}{P}\right]}_{V_{m+1}\text{col.comp}}, \end{aligned} \tag{25} $$

$$ {}^{(Sys)}T_{comm}(m, s, n, \varphi) \approx \underbrace{{}^{(Sys)}\tau_{comm}^{p>0,q>0}\left[2m(P-1)s^2\right]}_{\text{ortho.ph.: step 1}} + \underbrace{{}^{(Sys)}\tau_{comm}^{p=0,q>0}\left[(P+1)Ps^2\right]}_{\text{ortho.ph.: step 2}} + \underbrace{{}^{(Sys)}\tau_{comm}^{p>0,q>0}\left[2(P-1)s\frac{n}{P}\right]}_{V_{m+1}\text{col.comp}} \tag{26} $$

Variable φ represents the *sparsity* of matrix $\mathcal{A}$ and is defined as $\varphi = \frac{nnz(\mathcal{A})}{n^2}$ where $nnz(\mathcal{A})$ is the number of nonzero elements of $\mathcal{A}$. The symbols ${}^{(Sys)}\tau_{comp}^{p=0}$ and ${}^{(Sys)}\tau_{comp}^{p>0}$ respectively represent the time in seconds to perform a floating-point operation on tasks $p = 0$ and $p > 0$. The symbols ${}^{(Sys)}\tau_{comm}^{p=0,q>0}$ and ${}^{(Sys)}\tau_{comm}^{p>0,q>0}$ respectively represent (1) the time in seconds to transfer a floating-point number among tasks p and q where $p = 0$ and $q > 0$ and (2) the time in seconds to transfer a floating-point number among tasks p and q where $p > 0$ and $q > 0$. Such notations assume that

$$ \begin{aligned} {}^{(Sys)}\tau_{comp}^{p_1>0} &= {}^{(Sys)}\tau_{comp}^{p_2>0}, \quad \forall p_1, p_2 = 1, \dots, P, \\ {}^{(Sys)}\tau_{comm}^{p=0,q_1>0} &= {}^{(Sys)}\tau_{comm}^{p=0,q_2>0}, \quad \forall q_1, q_2 = 1, \dots, P, \\ {}^{(Sys)}\tau_{comm}^{p_1>0,q_1>0} &= {}^{(Sys)}\tau_{comm}^{p_2>0,q_2>0}, \quad \forall p_1, p_2, q_1, q_2 = 1, \dots, P. \end{aligned} $$

To write Equations (25) and (26) we consider that in Algorithm 6:

1. the matrix–matrix products at line 6 require $ms^2\frac{1}{n}$ floating-point operations,
2. the communication phase at line 7 executes $2(P-1)$ send/receive operations, during each operation ms^2 floating-point numbers are transferred,
3. the matrix–matrix products, combined with an AXPY operations[1], at line 8 require $ms^2\frac{1}{n} + s\frac{1}{n}$ floating-point operations,
4. the matrix–matrix products, combined with an AXPY operations, at line 11 require $ms^2\frac{1}{n} + s\frac{1}{n}$ floating operations and are executed $P-1$ times,
5. the AXPY operations at line 12 requires ms^2 floating-point operations and are executed $P-1$ times,
6. the computation of QR factorization at line 19 requires $2s^2\frac{1}{n}$ floating-point operations,[24]
7. the communication phase at line 21 executes P receive operations, during each operation s^2 floating-point numbers are transferred,
8. the computation of Q^l matrix at line 22 requires $O(Ps^3)$ floating-point operations that are executed only on task $p = 0$,
9. the communication phase at line 22 executes P send operations, during each operation Ps^2 floating-point numbers are transferred,
10. the matrix–matrix products at line 26 requires $Ps^2\frac{1}{n}$ floating-point operations,
11. the computation of $\underline{\mathfrak{H}}_{m+1}$ matrix at line 28 requires $O(m^3s^3)$ floating-point operations that are executed only on task $p = 0$.

We also consider that to compute the column of V_{m+1}^p at line 3 of the Algorithm 6 each task $p = 1, \dots, P$ has:

[1]The term AXPY indicates a generalized vector addition of the form $y = y + \alpha x$.

1. to compute s matrix–vector operations of the form $(\mathcal{A}_{p,j})_{j=1,\dots,P}(x_j)_{j=1,\dots,P}$ for a total of about $s\frac{nnz(\mathcal{A})}{P}$ floating-point operations.
2. to compute $(P-1)s$ AXPY operations for a total of about $(P-1)s\frac{n}{P}$ floating-point operations.
3. to execute $2(P-1)s$ send/receive operations, during each operation at most $\frac{n}{P}$ floating-point numbers are transferred.

To carry out a study of the behavior of granularity ${}^{(Sys)}G(m,s,n,\varphi)$ as the size of the problem n increases and $P=1$, as often happens with heterogeneous computing system mounting just one accelerator device to which are delegated the most computationally intensive phases, taking into account that $s, m \ll n$, some of the terms of Equation (25) can be overlooked. Furthermore, we can assume that ${}^{(Sys)}\tau_{comm}^{p>0,q>0} = 0$, both Equations (25) and (26) could be rewritten as

$$ {}^{(Sys)}T_{comp}(m,s,n,\varphi) \approx {}^{(Sys)}\tau_{comp}^{p>0}\frac{n}{P}\left\{s\varphi n + [(P+1)m + P + 2]s^2 + (2P-1)s + \frac{P(P-1)ms^2}{n}\right\}, \tag{27} $$

$$ {}^{(Sys)}T_{comm}(m,s,n,\varphi) \approx {}^{(Sys)}\tau_{comm}^{p=0,q>0}\{(P+1)Ps^2\}. \tag{28} $$

Then, taking into account the asymptotic behavior as the size of the problem n increases, Equation(24) for the considered algorithm (when $P=1$) could be written as:

$$ {}^{(Sys)}G(m,s,n,\varphi) \underset{\lim_{n\to\infty}}{\approx} \frac{{}^{(Sys)}\tau_{comp}^{p>0}}{{}^{(Sys)}\tau_{comm}^{p=0,q>0}} n\frac{s\varphi n + (m+3)s^2 + s}{2s^2} \underset{\lim_{n\to\infty}}{\approx} \underbrace{\frac{{}^{(Sys)}\tau_{comp}^{p>0}}{{}^{(Sys)}\tau_{comm}^{p=0,q>0}}}_{\text{Computing environment}} \underbrace{\frac{\varphi n^2}{2s}}_{\text{matrix sparsity and dimension}} = {}^{(Sys)}\zeta\,\kappa(m,s,n,\varphi), \tag{29} $$

where we pose

$$ {}^{(Sys)}\zeta = \frac{{}^{(Sys)}\tau_{comp}^{p>0}}{{}^{(Sys)}\tau_{comm}^{p=0,q>0}}, \tag{30} $$

$$ \kappa(m,s,n,\varphi) = \frac{\varphi n^2}{2s}. \tag{31} $$

From Equation (29) it is evident that ${}^{(Sys)}G(m,s,n,\varphi)$ depends on the characteristics of the computing environment, on the sparsity of $\mathcal{A}$, on the problem size n and on the value of s. Note that in the model considered for ${}^{(Sys)}G(m,s,n,\varphi)$ the overlap of computation and communication phases is not considered.

In order to "asses portability" of the proposed algorithm, the *performance portability* metric $\mathcal{P}(m,s,n,\varphi,H)$ which is defined in Pennycook et al.,[14] can be used:

$$ \mathcal{P}(m,s,n,\varphi,H) = \frac{|H|}{iH \in \sum \frac{1}{e_i(m,s,n,\varphi)}}, \tag{32} $$

where $e_i(m,s,n,\varphi)$ is defined as:

$$ e_i(m,s,n,\varphi) = \frac{{}^{(Sys)_i}G(m,s,n,\varphi)}{1 + {}^{(Sys)_i}G(m,s,n,\varphi)}. \tag{33} $$

It is easy to verify that:

- $e_i(m,s,n,\varphi)$ coincides with the fraction of the calculation time compared with the total time of execution of the considered algorithm implementations on the ith system;
- the bigger G, the more $e_i(m,s,n,\varphi)$ is close to one;
- for all $j \neq i$

$$ e_i(m,s,n,\varphi) < e_j(m,s,n,\varphi) \Leftrightarrow {}^{(Sys)_i}\zeta < {}^{(Sys)_j}\zeta; \tag{34} $$

- for all $\chi \in]0, \dots, 1[$

$$ e_i(m,s,n,\varphi) > \chi \Leftrightarrow \kappa(m,s,n,\varphi) > \frac{1}{{}^{(Sys)_i}\zeta}\left(1 - \frac{1}{\chi+1}\right). \tag{35} $$

The proposed *performance portability* metric is based on the harmonic mean of an application's performance efficiency observed across a set H of platforms. As discussed in Smith,[26] harmonic mean, unlike what is allowed by other types of mean such as arithmetic or geometric one, should be used for summarizing performance expressed as a rate.

5 | A CASE STUDY IN NUMERICAL LINEAR ALGEBRA

We show and discuss some performance tests related to the solution of (1) by Algorithm 6 where $\mathcal{A}$ is a structured banded matrix. We remember that the considered Algorithm is based on both BLAS2 and BLAS3 operations where the involved operands are both full and sparse (e.g., see respectively the QR factorization at line 19 and the matrix-power computations at line 3 of the Algorithm 6). Therefore, depending on the type of operand (a full or sparse matrix), the most suitable BLAS procedure is used.

In Computational Science, there are many examples of applications that lead to the use of band matrices: matrices from finite element or finite difference problems are often banded; other applications are based on matrices which can be successfully well approximated/reformulated by banded matrices (e.g., see Zhang et al.[27] and Genton[28] for some case studies in Machine Learning context). For these reasons banded matrices can be considered significative case studies.

To generate test banded matrices we use the *xLATMR* procedure from the LAPACK Test Suite[29] which can generate random dense/sparse by requesting, among others, the following actions: (1) generate a matrix $\mathcal{A}$ with random entries of a specified distribution, (2) make $\mathcal{A}$ a banded matrix, if desired, by zeroing out the matrix outside a band of bandwidth $2 * k$. In particular, by using the *xLATMR* procedure and for different values of matrix dimension n and different values of its sparsity φ, we generate the matrix $\mathcal{A}(n, \varphi)$ where $k \approx \frac{\varphi n - 1}{2}$. The considered values for φ should allow us to investigate a range of case studies whose matrices sweep from very scattered to almost dense matrices.

Tests are executed on three different systems whose hardware features are:

System 1 a Dell Inc. PowerEdge R720 server with:

- 2 Intel(R) 10-core Xeon(R) CPUs,
- 1 NVIDIA Tesla K20m.

The accelerator device is accessible by a PCI Express connection whose average Measured Bandwidth ${}^{(Sys)_1}\gamma_{CPU-GPU}$ is about 6.3 GB/s. Its Double Precision Performance ${}^{(Sys)_1}\tau_{GPU}$ is 572 GFlops. Therefore, ${}^{(Sys)_1}\zeta = 1.1014e-02$.

System 2 a Dell Inc. PowerEdge R510 server with:

- 2 Intel(R) quad-core Xeon(R) CPUs,
- 1 NVIDIA TITAN Xp.

The accelerator device is accessible by a PCI Express bus connection whose average Measured Bandwidth ${}^{(Sys)_2}\gamma_{CPU-GPU}$ is about 3.2 GB/s. Its Double Precision Performance ${}^{(Sys)_2}\tau_{GPU}$ is 395 GFlops. Therefore, ${}^{(Sys)_2}\zeta = 8.1013e-03$.

System 3 a Dell Inc. PowerEdge R7425 server with:

- 2 AMD(R) EPYC 7301 16-Core CPUs,
- 1 NVIDIA Tesla V100-PCIE-32GB.

The accelerator device is accessible by a PCI Express bus connection whose average Measured Bandwidth ${}^{(Sys)_3}\gamma_{CPU-GPU}$ is about 12.9 GB/s. Its Double Precision Performance ${}^{(Sys)_3}\tau_{GPU}$ is 5499 GFlops. Therefore, ${}^{(Sys)_3}\zeta = 2.3459e-03$.

In Table 1 we report some details about how the considered Hosts and Devices are connected: in particular the PCI Express (PCIe) generations of both the Host Bus and the Device Connector are listed. In the same table the Peak Bandwidths, as expected by the standards of the various PCIe generations,[30] are reported. We also report the number of lanes on which each PCI connection is based. The Measured Bandwidths ${}^{(Sys)_{(\cdot)}}\gamma_{CPU-GPU}$ seem to be consistent with the Peak Bandwidths reported in Table 1 also if some connection combinations seem to have a particularly negative influence on data transfer performances (i.e., see System 2).

All these systems, although homogeneous in the architecture that combines multicore CPUs with just one GPU, are quite different for both computation and communication performances. The tasks $p = 0, \ldots, 1$, used to execute the Algorithm 6, are mapped to CPUs (where $p = 0$) and GPUs (where $p = 1$).

TABLE 1 Details of Host-Device PCIe connections in systems used for tests executions

	Generation of host bus	# of lanes of host bus	Peak bandwidth of host bus (GB/s)	Generation of device	# of lanes of device	Peak bandwidth of device (GB/s)
System 1	3.0	16	15.75	2.0	16	8.0
System 2	2.0	8	4.0	3.0	16	15.75
System 3	3.0	16	15.75	3.0	16	15.75

On all the systems are available the following software tools:

- **System 1** release 6.7 of a Scientific Linux distribution,
  System 2 release 16.04.4 of a Ubuntu Linux distribution,
  System 3 release 7.7.1908 of a CentOS Linux distribution,
- version 10.2 of the Nvidia CUDA Toolkit package,
- version 11.1.3 of the Intel MKL library,
- version 2.5.1 of the MAGMA package.

The value of ${}^{(Sys)_i}\tau_{GPU}$ is evaluated by some executions of the MAGMA dGEMM procedure. In the same manner, the value of ${}^{(Sys)_i}\gamma_{CPU-GPU}$ is evaluated by some executions of the `bandwidthTest` from CUDA Toolkit when the *pinned memory* is used. A schematic representation of the environment, as shown by the MAGMA `magma_print_environment()` function, is listed in Figure 2.

We note that the considered Algorithm is based on both BLAS2 and BLAS3 operations where the involved operands are both full and sparse (e.g., see respectively the QR factorization at line 19 and the matrix-power computations at line 3 of the Algorithm 6). Therefore, depending on the type of operand (a full or sparse matrix), the most suitable MAGMA procedure is used.

In Figures 3-6 are showed some results related to the execution times (both total and related to each phase of computation for all the three considered systems) of the Algorithm 6 as functions of problem dimension n for different values s. For all the considered values of n, different values for the sparsity of matrix $\mathcal{A}$ are considered. Tests have been scheduled on the considered systems taking into account the availability of RAM on the GPU devices (see Table 2 for details). Tests with higher memory requirements (i.e., with higher values for φ) are scheduled just on the second and the third systems.

From the analysis of the times shown in the aforecited figures, it is evident that the total execution time (see plots (*).1) is dominated by the time for the calculation of the matrix powers (see the yellow boxes in plots (*).2). This seems consistent with the ideal model for G (as defined in Equation (29)).

In Figure 7 the trends of $e_i(m, s, n, \varphi)$ performance metrics, as functions of problem dimension n, for different values s, different values for sparsity φ computed on all the considered systems, are plotted: lines with the same color is related to the same value of s; lines related to each system are identified by the same symbol: the symbols ●, ▲, and ♦ tag respectively the systems 1, 2, and 3. In Figure 8 the trends of $\mathcal{P}(m, s, n, \varphi, H)$ *performance portability* metric, as functions of problem dimension n, for different values s and different values for sparsity φ, are plotted: as above described lines with the same color is related to the same value of s. $\mathcal{P}(m, s, n, \varphi, H)$ metrics, depending on the values of φ, are computed on the two different Systems Sets $H^{\natural}$ and $H^{\dagger}$.

System 1

```
% MAGMA 2.5.1  compiled for CUDA capability >= 3.0.
% CUDA runtime 10020, driver 10020. OpenMP threads 20. MKL 11.1.3, MKL threads 20.
% device 0: Tesla K20m, 705.5 MHz clock, 4743.9 MiB memory, capability 3.5
% cpu 0-1: Intel(R) Xeon(R) E5-2680 v2 @ 2.80GHz
```

System 2

```
% MAGMA 2.5.1  compiled for CUDA capability >= 3.0.
% CUDA runtime 10020, driver 10020. OpenMP threads 8. MKL 11.1.3, MKL threads 8.
% device 0: TITAN Xp, 1582.0 MHz clock, 12196.1 MiB memory, capability 6.1
% cpu 0-1: Intel(R) Xeon(R) E5506 @ 2.13GHz
```

System 3

```
% MAGMA 2.5.1  compiled for CUDA capability >= 3.0.
% CUDA runtime 10020, driver 10020. OpenMP threads 64. MKL 11.1.3, MKL threads 32.
% device 0: Tesla V100-PCIE-32GB, 1380.0 MHz clock, 32510.5 MiB memory, capability 7.0
% cpu 0-1: AMD(R) EPYC(R) 7301 @ 2.19GHz
```

FIGURE 2 Description of the computing environments used by performance tests

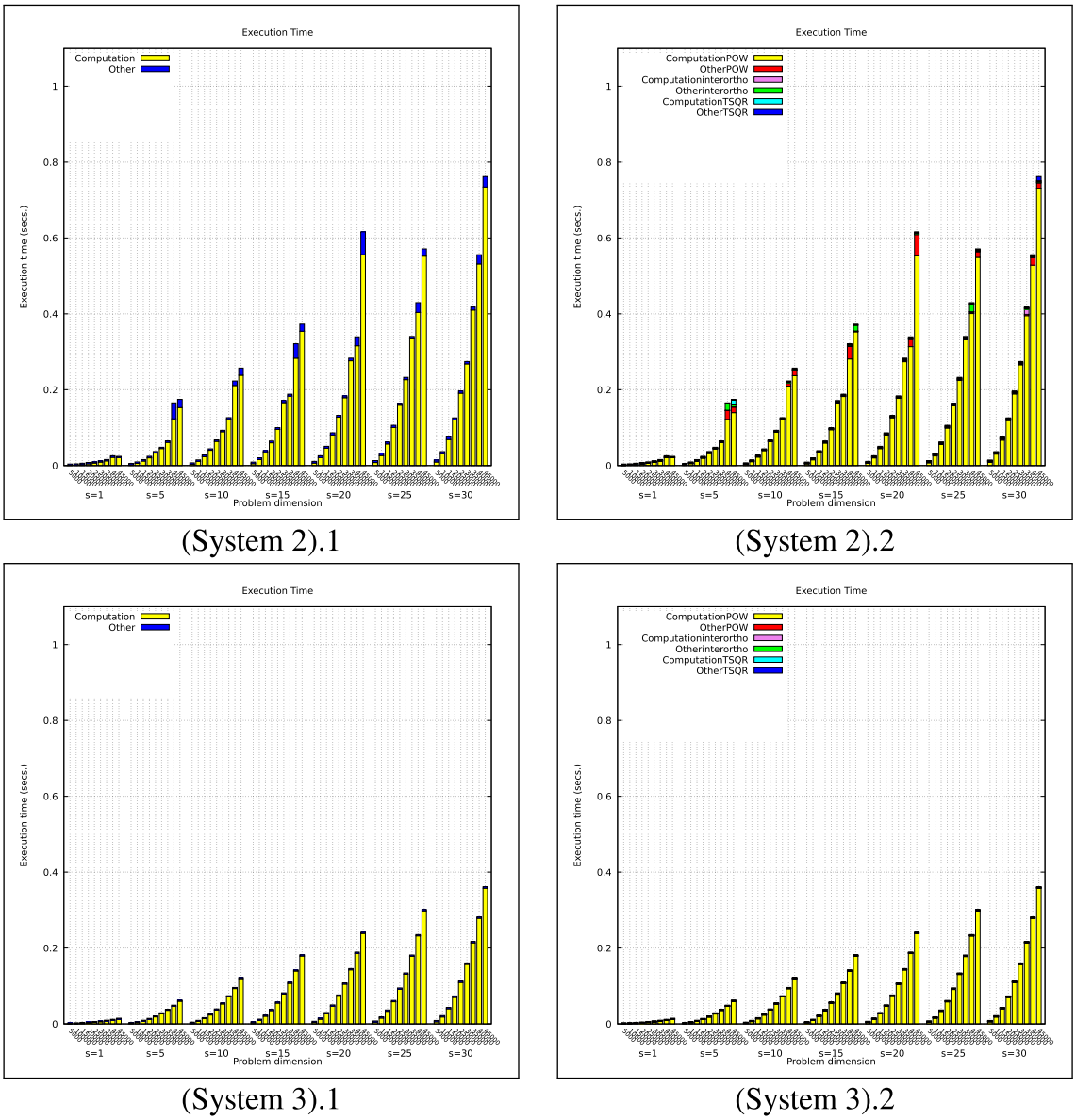

FIGURE 3 Total Execution time (*).1 and Execution times for each phase (*).2 of the Algorithm 6 as functions of problem dimension n for different values s. The value of matrix sparsity is $\varphi = 0.40$

From the Figure 7 we can observe that:

- given the computational complexity $\kappa(m, s, n, \varphi)$ of the problem, the measured value for $e_i(m, s, n, \varphi)$ is as small as smaller is ${}^{(Sys)_i}\zeta$ consistently with inequality (34), for example, see the violet lines ($s = 1$) related to $\varphi = 0.20$;
- given the computing system (i.e., ${}^{(Sys)_i}\zeta$), to get a "good" value for $e_i(m, s, n, \varphi)$ (say at least χ) it is necessary that $\kappa(m, s, n, \varphi)$ is as much larger as smaller is ${}^{(Sys)_i}\zeta$: e.g., if $s = 1$, $\varphi = 0.20$ and $\chi = 0.5$, on System 1 all the considered values for n are suitable while on System 3 should be $n \geq 35,000$. All that is consistent with the asymptotic behavior described by inequality (35).

All the plots in Figure 8 confirm that, since $\mathcal{P}(m, s, n, \varphi, H)$ is a suitable mean "dominated by the minimum"[2]of $e_i(m, s, n, \varphi)$ values,[31] for this to take "good" values (say at least 0.5) it is necessary that all of $e_i(m, s, n, \varphi)$ values are good enough and this happens only when the computational complexity $\kappa(m, s, n, \varphi)$ of the problem is sufficiently large.

[2] If $H(x_1, \ldots, x_n)$ is the Harmonic Mean of n positive values, then $\min(x_1, \ldots, x_n) \leq H(x_1, \ldots, x_n) \leq n \min(x_1, \ldots, x_n)$.

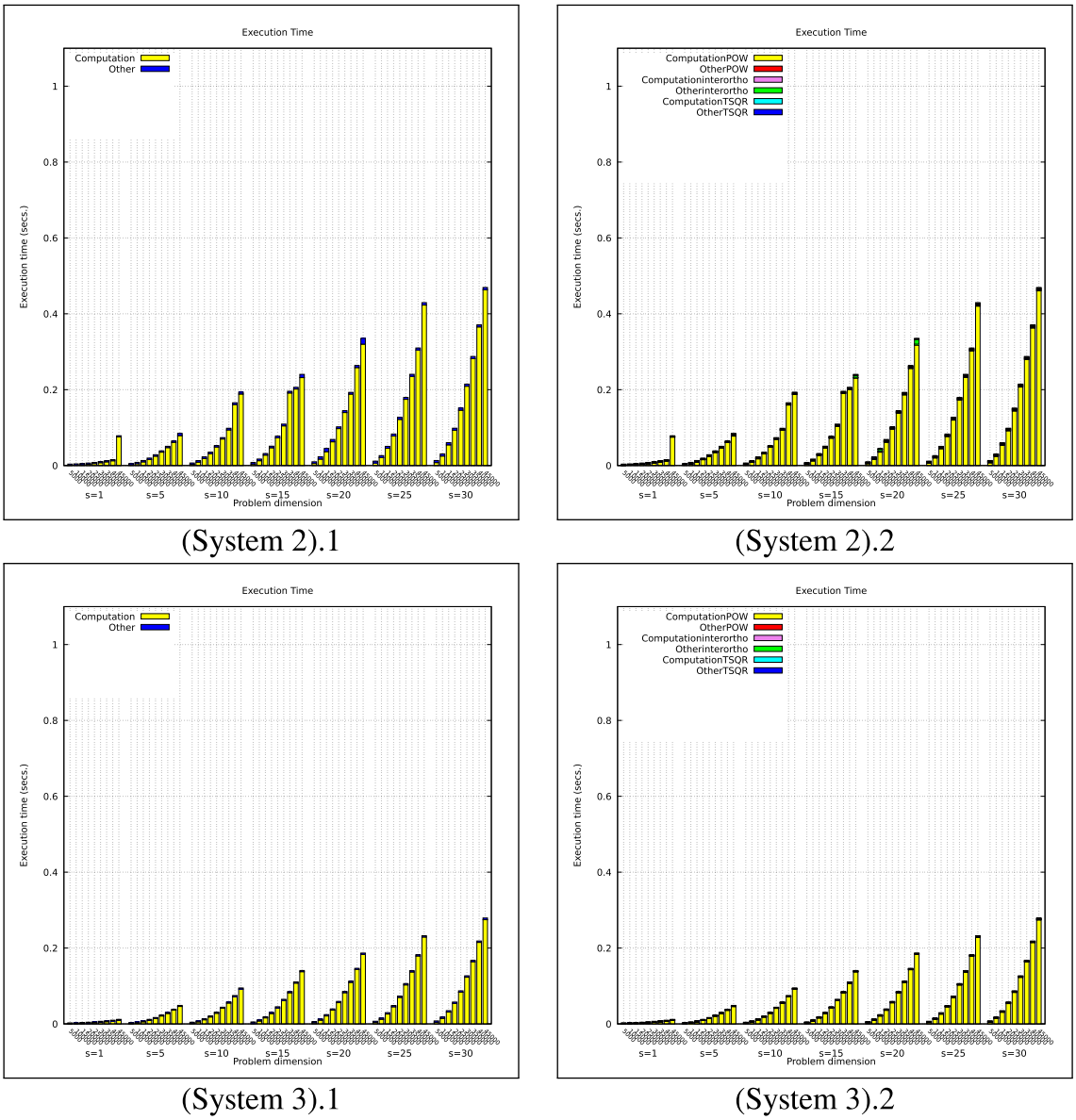

FIGURE 4 Total Execution time (*).1 and Execution times for each phase (*).2 of the Algorithm 6 as functions of problem dimension n for different values s. The value of matrix sparsity is $\varphi = 0.30$

6 | CONCLUSIONS AND FUTURE WORK

The work proposes novel parallel revisitation of the modules, used in block-based Krylov iteration methods, which are based on the overlapping of communication and computation. For such revisitation, we gave a model of their granularity, in order to evaluate the *performance portability*, and verified in a heterogeneous computing environment the theoretical results for a case study related to a classical problem from numerical linear algebra. Other case studies from different contexts as Chemistry and Material Science, Data Analysis and Machine Learning (ML), Image Processing should be considered in our future work. Scientific and engineering responses from all these contexts seem to be critical to virtually every the United Nations Sustainable Development Goals such as "climate action" and "good health and well-being."[32]

Imaging technologies (e.g., very large telescopes, medical imaging scanners, and modern microscopes) are based on devices that collect electromagnetic energy connected to computing systems that assemble the collected data into images. The "assembling" process typically involves solving an inverse problem, that is, the image is reconstructed from indirect measurements of the corresponding object. Inverse problems are ubiquitous in imaging applications,[33] including the images deblurring and reconstruction processes,[34-38] and are very often based on the solution of linear systems like those described by Equation (1). These problems typically require processing a large amount of data (the number of pixels or voxels in the discretized image) and the related equation's systems have a very large number of equations (e.g., $O(10^9)$ for a 3D image reconstruction problem).

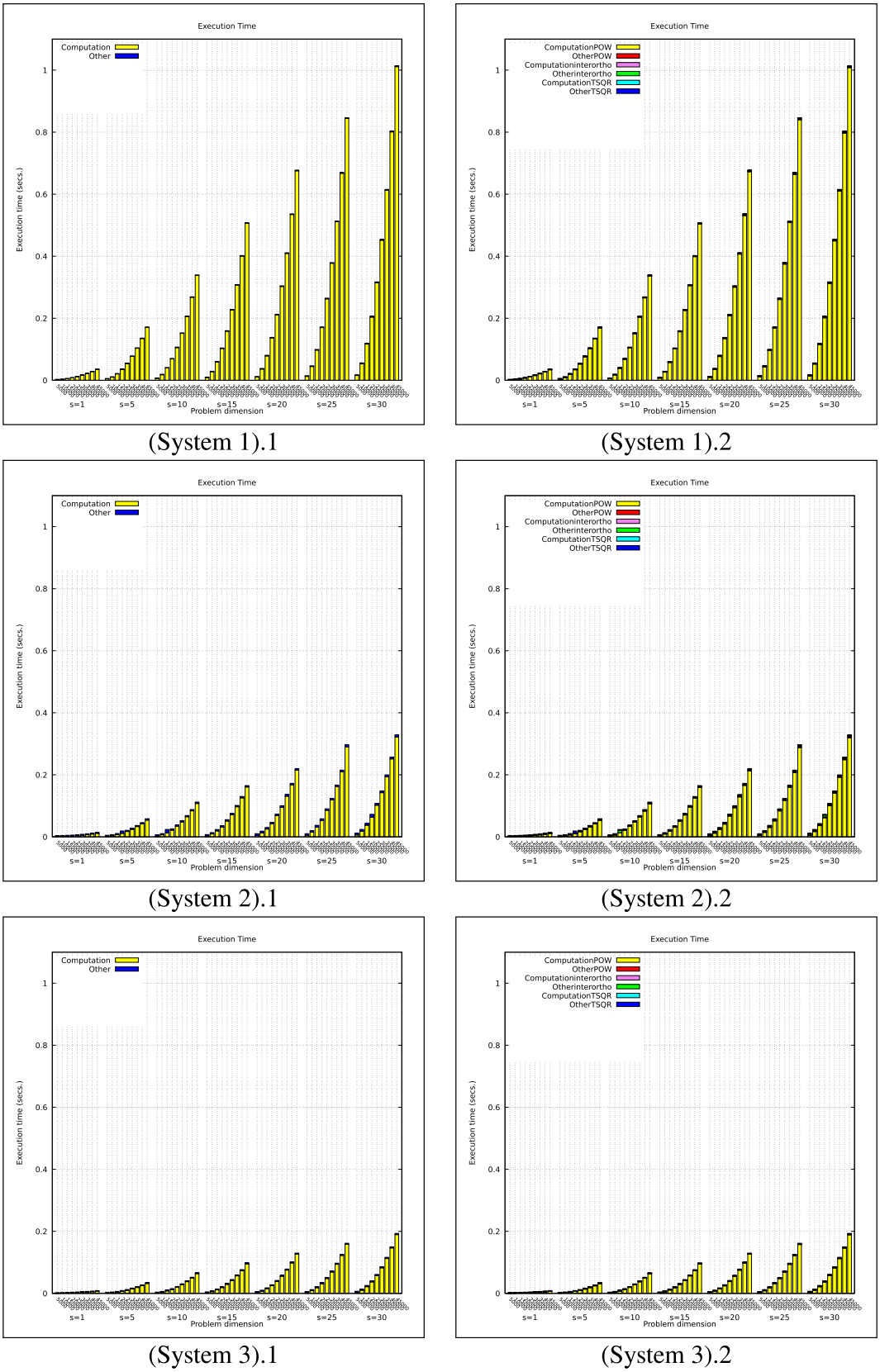

FIGURE 5 Total Execution time (*).1 and Execution times for each phase (*).2 of the Algorithm 6 as functions of problem dimension n for different values s. The value of matrix sparsity is $\varphi = 0.20$

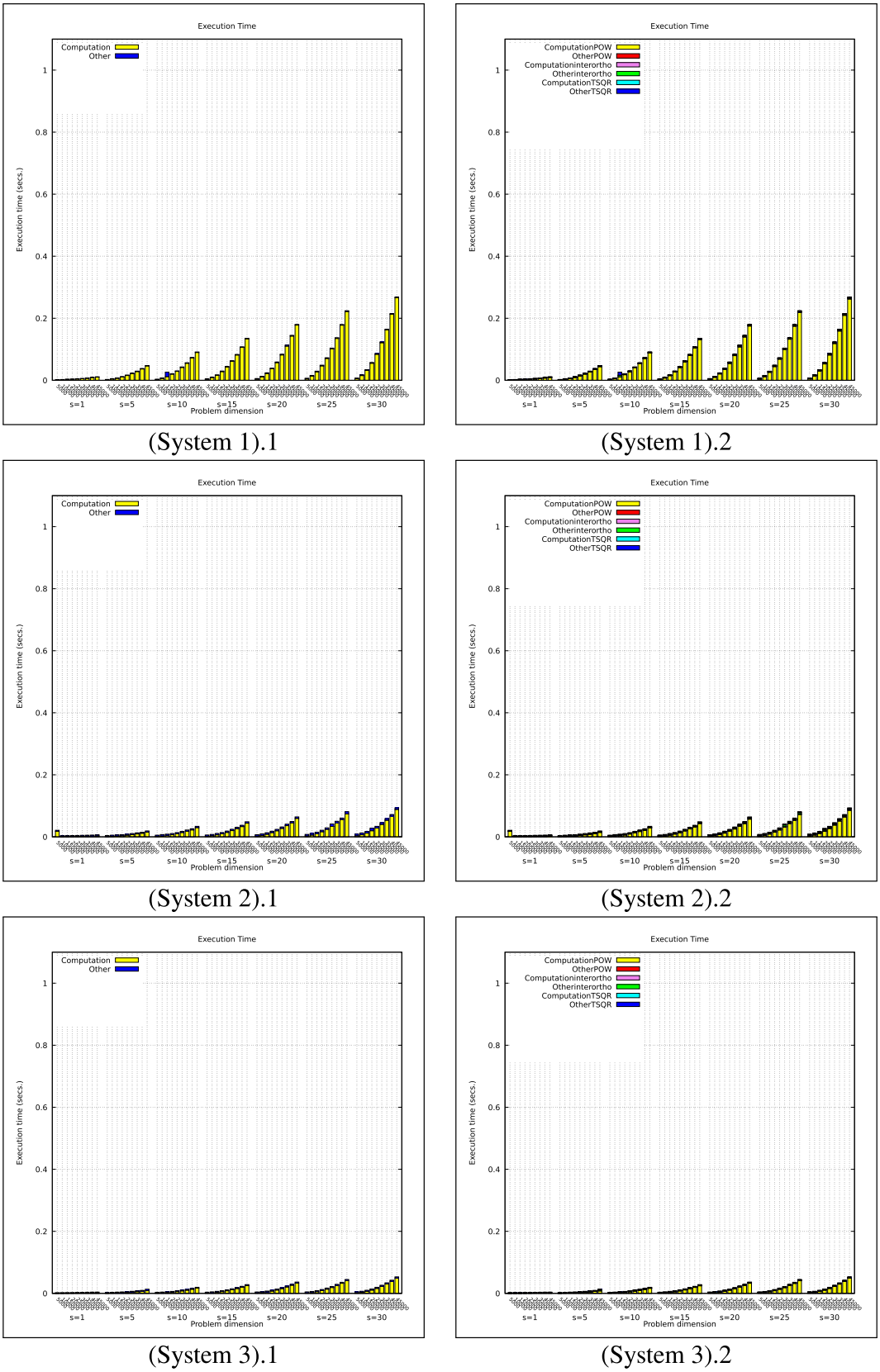

FIGURE 6 Total Execution time (*).1 and Execution times for each phase (*).2 of Algorithm 6 as functions of problem dimension n for different values s. The value of matrix sparsity is $\varphi = 0.05$

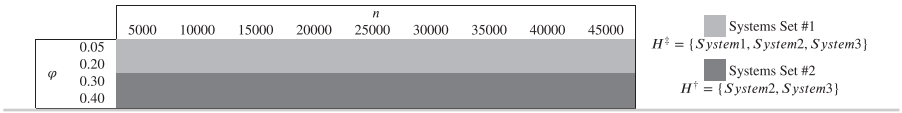

TABLE 2 Scheduling of the tests on the considered systems

If, as often happens, these problems must be solved in real-time (e.g., see X-ray spectroscopy applications described in Reference 39), the use of particularly efficient algorithms becomes mandatory.

Computer simulation of phenomena in the contest of the Chemistry and of the Material Science give us the opportunity of a glance on the structure of the matter providing, thanks to the most suitable mathematical models, information both on microscopic details (the masses of the atoms, the interactions between them, molecular geometry, etc.) and on its macroscopic behavior. Depending on the scale of simulation (ranging from the continuum to quantum scale), and then from the used mathematical models, the computational kernel of the simulation process could be the solution of linear systems[40] or the solution of sparse eigenvalue problems.[41] In both cases, the algorithms based on KM methods such as respectively the GMRES and the Lanczos iteration, and all their communication-avoiding variants,[8] could be exploited to perform simulations whose computational cost has been up to now prohibitive.

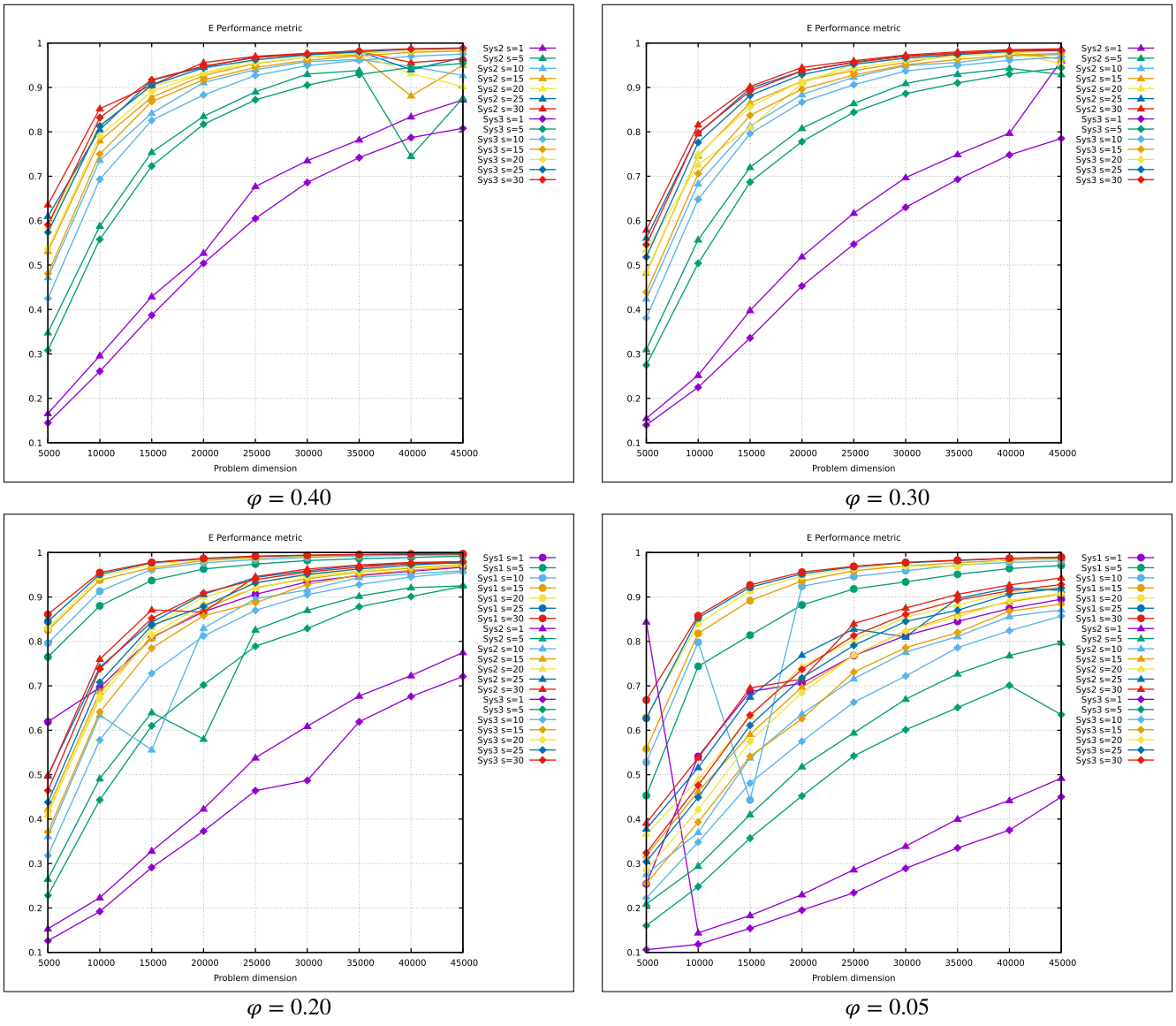

FIGURE 7 Trends of *e* performance metrics of Algorithm 6 as functions of problem dimension *n* for different values *s* and different values for sparsity φ of matrix $\mathcal{A}$

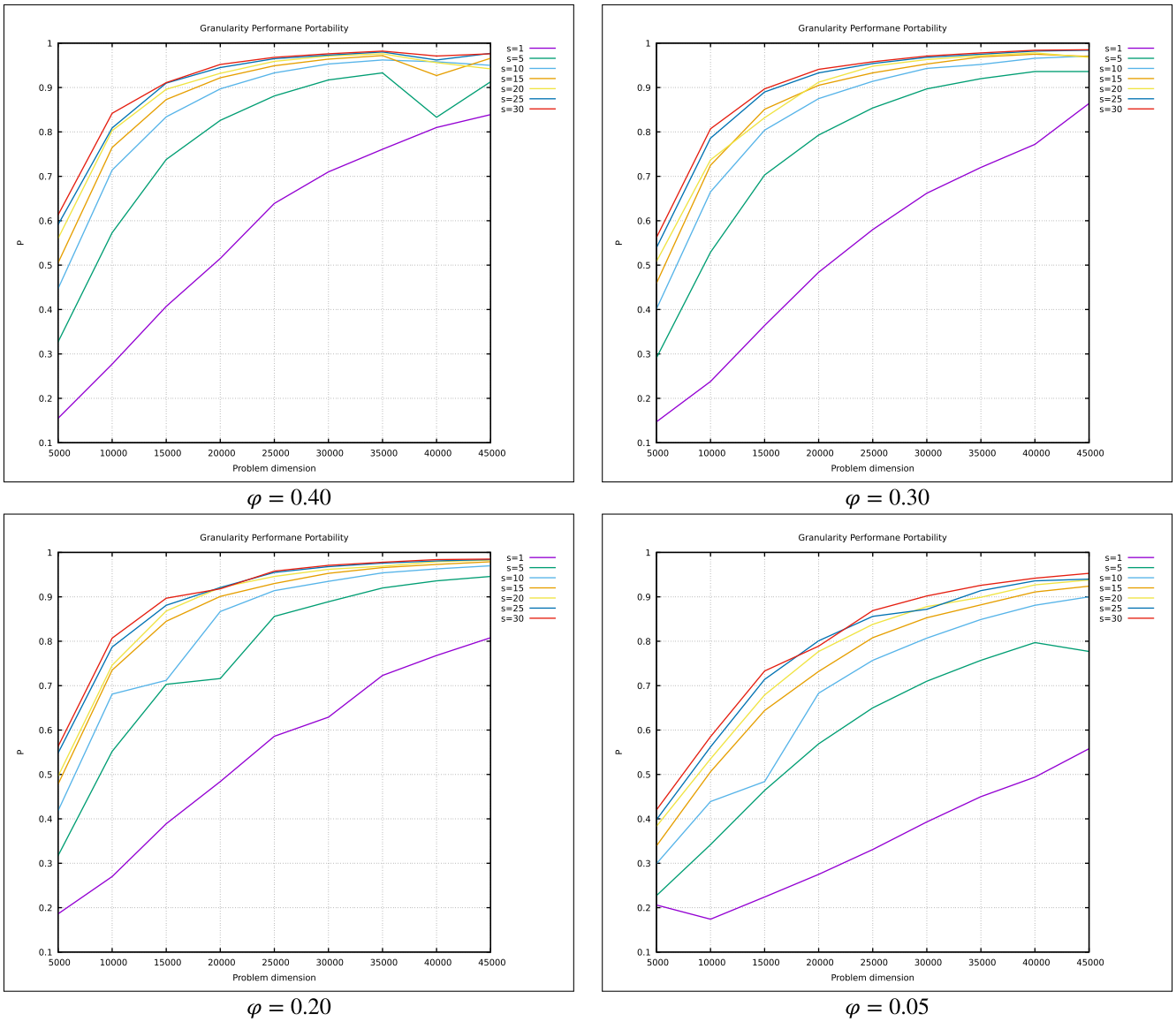

FIGURE 8 Granularity Portability $\mathcal{P}$ of Algorithm 6 as functions of problem dimension n for different values s and different values for sparsity φ of matrix $\mathcal{A}$

The need to automate processes to analyze data is at the moment mandatory considered the huge amount of data that continues to increase. The availability of efficient methods and tools able to automate such process, and to learn from experience, could help in obtaining knowledge from data, that is, to understand speech or images, make diagnoses in medicine and support basic scientific research. Machine learning algorithms usually require a high amount of numerical computation. Common operations include optimization (finding the value of an argument that minimizes or maximizes a function using iterative methods that update estimates of the solution via an iterative process) and solving systems of linear equations.[27,28,42] Therefore, it could be worthy of attention to verify whether the use of BKM can help in improving the effectiveness of learning methods in an attempt to bring them ever closer to the same human mechanisms.

For all the above described applications, the use of considered KM block-based algorithm could be useful to reduce the problem's "time to solution" exploiting at the best the hybrid and heterogeneous architecture of the new exascale computing systems.

From all the proposed tests results we can conclude that:

1. to appreciate the improvement in granularity (and granularity portability) of the implementation of Algorithm 6 we should have (as already shown by the model) that the number of nonzeros elements of $\mathcal{A}$ is high enough respect to the computation performance potentially expressible by the computing system;
2. as promised, higher values of s, guarantee a high level of granularity (and granularity portability) especially for higher values n of problem dimension.

For all that has been said about the predominance of the time spent on matrix power computations, it can be concluded that the improvement of the granularity of the KM block-based methods depends on the quality of algorithms used for the aforementioned operation. These algorithms, which could for example be inspired by those described in Mohiyuddin et al.,[10] must however take into account the possible need to use a preconditioner as often happens when it is necessary to improve the convergence speed of the iterative methods.

Our future works intends:

- to invest some effort to develop new algorithms able to improve granularity (e.g., algorithms for the matrix-power kernel),
- to elaborate further on the issue related to the evaluation of the performance portability of KM block-based algorithms on the computing systems which will respond to the new requirements of the incoming exascale era when computing environments will include very large MIMD systems, heterogeneous CPU-GPU systems and combinations of both all equipped with standard scientific libraries as PETSc[43] and MAGMA[12] and
- to investigate the *performance portability* evaluation in a more general performance evaluation framework we developed and already applied in a few other works[18,19,44-46] and we intend to continue to evolve.

ACKNOWLEDGMENT

This work used the infrastructure provided to the SCoPE Data Centre[47] of Naples "Federico II" also by project IBISCO, code PIR01_00011, PON 2014-2020.

ORCID

Luisa Carracciuolo https://orcid.org/0000-0002-8521-1645
Valeria Mele https://orcid.org/0000-0002-2643-3483
Lukasz Szustak https://orcid.org/0000-0001-7429-6981

REFERENCES

1. Dongarra J, Sullivan F. Guest editors' introduction: the top 10 algorithms. *Comput Sci Eng*. 2000;2(1):22-23. https://doi.org/10.1109/MCISE.2000.814652.
2. The Production-ready Exascale-enabled Krylov solvers for exascale computing (PEEKS) project web page; 2019. https://icl.utk.edu/peeks/ [Online; accessed May 20, 2020].
3. The Exascale Computing Project Website. https://www.exascaleproject.org/ [Online; accessed May 20, 2020].
4. Yamazaki I, Hoemmen M, Luszczek P, Dongarra J. Improving performance of gmres by reducing communication and pipelining global collectives. Paper presented at: Proceedings of the 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lake Buena Vista, FL; 2017:1118-1127. https://doi.org/10.1109/IPDPSW.2017.65.
5. Lawson CL, Hanson RJ, Kincaid DR, Krogh FT. Basic linear algebra subprograms for Fortran usage. *ACM Trans Math Softw*. 1979;5(3):308-323. https://doi.org/10.1145/355841.355847.
6. Kwiatkowski J. Evaluation of parallel programs by measurement of its granularity. *Proc Parall Process Appl Math*. 2002;2328:145-153. https://doi.org/10.1007/3-540-48086-2_16.
7. Bai Z, Hu D, Reichel L. A Newton basis GMRES implementation. *IMA J Numer Anal*. 1994;14(4):563-581. https://doi.org/10.1093/imanum/14.4.563.
8. Hoemmen M. Communication-avoiding Krylov Subspace Methods [PhD thesis]. University of California at Berkeley, Berkeley, CA; 2010:AAI3413388.
9. Ghysels P, Ashby T, Meerbergen K, Vanroose W. Hiding global communication latency in the gmres algorithm on massively parallel machines. *SIAM J Sci Comput*. 2013;35(1):C48-C71. https://doi.org/10.1137/12086563X.
10. Mohiyuddin M, Hoemmen M, Demmel J, Yelick K. minimizing communication in sparse matrix solvers. Paper presented at: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, Portland, Oregon; 2009:36:1-36:12. https://doi.org/10.1145/1654059.1654096.
11. Imberti D, Erhel J. Varying the s in your s-step GMRES. *Electron Trans Numer Anal (ETNA)*. 2017;47:206-230.
12. The matrix algebra on GPU and multicore architecture (MAGMA) library website. http://icl.cs.utk.edu/magma/ [Online; accessed May 20, 2020].
13. The scalable LAPACK project. http://www.netlib.org/scalapack/ [Online; accessed May 20, 2020].
14. Pennycook SJ, Sewall JD, Lee VW. Implications of a metric for performance portability. *Future Generat Comput Syst*. 2017;92:947–958. https://doi.org/10.1016/j.future.2017.08.007.
15. *DOE Centres of Excellence Performance Portability Meeting: Post-meeting Report Technical Report LLNL-TR-700962*. Livermore: Lawrence Livermore National Laboratory; 2016. https://asc.llnl.gov/sites/asc/files/2020-09/COE-PP-Meeting-2016-FinalReport_0.pdf.
16. Laccetti G, Montella R, Palmieri C, Pelliccia V. The high performance Internet of Things: using GVirtuS to share high-end GPUs with ARM based cluster computing nodes. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) LNCS vol. 8384 (PART 1); 2014:734-744. https://doi.org/10.1007/978-3-642-55224-3_69.
17. Carracciuolo L, D'Amore L, Mele V. Toward a fully parallel multigrid in time algorithm in PETSc environment: a case study in ocean models. Paper presented at: Proceedings of the 2015 International Conference on High Performance Computing Simulation (HPCS), Amsterdam, Netherlands; 2015:595-598. https://doi.org/10.1109/HPCSim.2015.7237098.
18. Mele V, Constantinescu EM, Carracciuolo L, D'amore L. A PETSc parallel-in-time solver based on MGRIT algorithm. *Concurr Comput Pract Exp*. 2018;30(24):4928. https://doi.org/10.1002/cpe.4928.

19. Mele V, Romano D, Constantinescu EM, Carracciuolo L, D'Amore L. Performance evaluation for a PETSc parallel-in-time solver based on the MGRIT algorithm. Paper presented at: Proceedings of the Euro-Par 2018: Parallel Processing Workshops, Turin, Italy; 2019:716-728. https://doi.org/10.1007/978-3-030-10549-5_56.
20. Future and emerging technologies: work part of the European horizon 2020 program. http://ec.europa.eu/research/participants/data/ref/h2020/wp/2016_2017/main/h2020-wp1617-fet_en.pdf [Online; accessed May 20, 2020].
21. Carracciuolo L, Lapegna M. Implementation of a non-linear solver on heterogeneous architectures. *Concurr Comput Pract Exp*. 2018;30(24):4903. https://doi.org/10.1002/cpe.4903.
22. Saad Y. *Iterative Methods for Sparse Linear Systems*. 2nd ed. Filadelfia, USA: Society for Industrial and Applied Mathematics; 2003.
23. Rosendale JV. Minimizing inner product data dependencies in conjugate gradient iteration. Proceedings of the International Conference on Parallel Processing (ICPP); 1983:44-46. https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19830026339.pdf.
24. Golub GH, Van Loan CF. *Matrix Computations*. 4th ed. Baltimora: JHU Press; 2013.
25. Çatalyürek ÜV, Aykanat C, Uçar B. On two-dimensional sparse matrix partitioning: models, methods, and a recipe. *SIAM J Sci Comput*. 2010;32(2):656-683. https://doi.org/10.1137/080737770.
26. Smith JE. Characterizing computer performance with a single number. *Commun ACM*. 1988;31(10):1202-1206. https://doi.org/10.1145/63039.63043.
27. Zhang HH, Genton M, Liu P. *Compactly Supported Radial Basis Function Kernels Institute of Statistics Mimeo Series No. 2570*. Raleigh, USA: , Department of Statistics of North Carolina State University; 2004. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.73.3200.
28. Genton MG. Classes of kernels for machine learning: a statistics perspective. *J Mach Learn Res*. 2002;2:299-312.
29. XLATMR: random matrices from the LAPACK test suite documentation page; 2019. https://math.nist.gov/MatrixMarket/data/misc/xlatmr/xlatmr.html [Online; accessed May 20, 2020].
30. PCI Express Wikipedia page. https://en.wikipedia.org/wiki/PCI_Express [Online; accessed May 20, 2020].
31. Harmonic mean Wikipedia page. https://en.wikipedia.org/wiki/Harmonic_mean [Online; accessed May 20, 2020].
32. European Commission Orientations towards the first strategic plan for Horizon Europe; 2019. https://ec.europa.eu/info/sites/info/files/research_and_innovation/strategy_on_research_and_innovation/documents/ec_rtd_orientations-he-strategic-plan_122019.pdf.
33. Chung J, Knepper S, Nagy JG. *Large-Scale Inverse Problems in Imaging*. New York, NY: Springer; 2015:47-90.
34. Antonelli L, Carracciuolo L, D'Amore L, Murli A. MEDITOMO: an high performance software for SPECT imaging. *Int J Comput Math*. 2009;86:31-56. https://doi.org/10.1080/00207160701504113.
35. Murli A, D'Amore L, Carracciuolo L, Ceccarelli M, Antonelli L. High performance edge-preserving regularization in 3D SPECT imaging. *Parall Comput*. 2008;34:115-132. https://doi.org/10.1016/j.parco.2007.12.004.
36. Carracciuolo L, D'Amore L, Murli A. Towards a parallel component for imaging in PETSc programming environment: a case study in 3-D echocardiography. *Parall Comput*. 2006;32:67-83. https://doi.org/10.1016/j.parco.2005.09.001.
37. Montella R, Kosta S, Oro D, et al. Accelerating linux and android applications on low-power devices through remote GPGPU offloading. *Concurr Comput Pract Exp*. 2017;29(24):e4286. https://doi.org/10.1002/cpe.4286.
38. Marcellino L, Montella R, Kosta S, et al. Using GPGPU accelerated interpolation algorithms for marine bathymetry processing with on-premises and cloud based computational resources. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), LNCS; vol. 10778, 2018:14-24. https://doi.org/10.1007/978-3-319-78054-2_2.
39. Siegel A., Draeger E., Deslippe J., et al. *Early application results on pre-exascale architecture with analysis of performance challenges and projections Exascale Computing Project (ECP) WBS 2.2 Milestone Report PM-AD-1080*. Springfield, USA: US Department of Energy; 2020. https://www.exascaleproject.org/wp-content/uploads/2020/03/ECP_AD_Milestone-Early-Application-Results_v1.0_20200325_FINAL.pdf.
40. Carracciuolo L, Casaburi D, D'Amore L, D'Avino G, Maffettone PL, Murli A. Computational simulations of 3D large-scale time-dependent viscoelastic flows in high performance computing environment. *J Non-Newtonian Fluid Mech*. 2011;166(23):1382-1395. https://doi.org/10.1016/j.jnnfm.2011.08.014.
41. Saad Y, Chelikowsky JR, Shontz SM. Numerical methods for electronic structure calculations of materials. *SIAM Rev*. 2010;52(1):3-54. https://doi.org/10.1137/060651653.
42. Goodfellow Ian, Bengio Yoshua, Courville Aaron. *Deep Learning*. Cambridge, USA: MIT Press; 2016. http://www.deeplearningbook.org.
43. Balay S, Abhyankar S, Adams MF, et al. PETSc Web page; 2019. http://www.mcs.anl.gov/petsc [Online; accessed May 20, 2020].
44. D'Amore L, Mele V, Laccetti G, Murli A. Mathematical approach to the performance evaluation of matrix multiply algorithm. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) LNCS; vol. 9574, 2016:25-34. https://doi.org/10.1007/978-3-319-32152-3_3.
45. Arcucci R, D'Amore L, Mele V. Mathematical approach to the performance evaluation of three dimensional variational data assimilation. *AIP Conf Proc*. 2017;1836(1):020001. https://doi.org/10.1063/1.4981941.
46. D'Amore L, Mele V, Romano D, Laccetti G. Multilevel algebraic approach for performance analysis of parallel algorithms. *Comput Inform*. 2019;38(4):817–850. https://doi.org/10.31577/cai_2019_4_817.
47. The SCoPE computing infrastructure website. http://www.scope.unina.it/ [Online; accessed May 20, 2020].

How to cite this article: Carracciuolo L, Mele V, Szustak L. About the granularity portability of block-based Krylov methods in heterogeneous computing environments. *Concurrency Computat Pract Exper*. 2021;33:e6008. https://doi.org/10.1002/cpe.6008