Computing and Informatics, Vol. 38, 2019, 817–850, doi: 10.31577/cai_2019_4_817

MULTILEVEL ALGEBRAIC APPROACH FOR PERFORMANCE ANALYSIS OF PARALLEL ALGORITHMS

Luisa D'Amore, Valeria Mele

University of Naples Federico II
Naples, Italy
e-mail: {luisa.damore, valeria.mele}@unina.it

Diego Romano

Institute of High Performance Computing and Networking (ICAR), CNR
Naples, Italy
e-mail: diego.romano@cnr.it

Giuliano Laccetti

University of Naples Federico II
Naples, Italy
e-mail: giuliano.laccetti@unina.it

Abstract. In order to solve a problem in parallel we need to undertake the fundamental step of splitting the computational tasks into parts, i.e. decomposing the problem solving. A whatever decomposition does not necessarily lead to a parallel algorithm with the highest performance. This topic is even more important when complex parallel algorithms must be developed for hybrid or heterogeneous architectures. We present an innovative approach which starts from a problem decomposition into parts (sub-problems). These parts will be regarded as elements of an algebraic structure and will be related to each other according to a suitably defined dependency relationship. The main outcome of such framework is to define a set of block matrices (dependency, decomposition, memory accesses and execution) which simply highlight fundamental characteristics of the correspond-

ing algorithm, such as inherent parallelism and sources of overheads. We provide a mathematical formulation of this approach, and we perform a feasibility analysis for the performance of a parallel algorithm in terms of its time complexity and scalability. We compare our results with standard expressions of speed up, efficiency, overhead, and so on. Finally, we show how the multilevel structure of this framework eases the choice of the abstraction level (both for the problem decomposition and for the algorithm description) in order to determine the granularity of the tasks within the performance analysis. This feature is helpful to better understand the mapping of parallel algorithms on novel hybrid and heterogeneous architectures.

Keywords: Complexity and performance of numerical algorithms, performance metrics, data decomposition, concurrency, parallel algorithms

Mathematics Subject Classification 2010: 65Y05, 65Y20, 68R01

1 INTRODUCTION AND MOTIVATION

Numerical algorithms are at the heart of the software that enable scientific discoveries. The development of effective algorithms has a tremendous impact on harnessing emerging computer architectures to achieve new science. The mapping problem, first considered in 1980s [8], refers to the implementation of algorithms on a given target architecture which is capable to maximize some performance metrics [5, 6, 31, 26, 32]. Due to the multidimensional heterogeneity of modern architectures, it is becoming increasingly clear that using the performance metrics in a one-size-fits-all approach fails to discover sources of performance degradation that hamper to deliver the desired performance level. The present article attempts to collect our efforts towards the development of a performance model, based on mathematical tools, guiding the understanding of computational tasks within an algorithm. We briefly summarize main novelties we provide in this work.

- We address the study of data dependencies in an algorithm, through the *dependency matrix*.
- We introduce the *decomposition matrix* describing a decomposition of the problem.
- We introduce the *execution matrix* describing the mapping of the algorithm on the computing machine.
- We define the *memory accesses matrix*, that helps us to define the software execution time.
- The block structure of the above matrices corresponds to the multiple levels (of the performance analysis) for the proposed approach. This feature is helpful for understanding the mapping of complex parallel algorithms solving real problems on novel hybrid and heterogeneous architectures.

- A set of parameters characterizing the matrices structure, namely their number of rows and columns, and a set of computing environment parameters, such as the execution time for one floating point operation, are used both to describe the problem and to compute speed up, efficiency, cost, overhead, scale up and operating point of the algorithm, *starting from the problem decomposition.*
- Even though for simplicity of notations we assume that at each level of parallelism the computing architecture is homogeneous, it is possible to extend the proposed framework considering – at the same level of decomposition – features of distributed algorithms on heterogeneous computing architectures. In this case, dependency matrix should firstly be employed to analyze problem and data decomposition; then, execution matrix, whose rows depend on the execution time of the machine operations at a given level of granularity, highlights the corresponding workload distribution at this level (cfr. Section 6).

The article is organized as follows. Section 2 will review basic concepts and definitions useful for setting up the mathematical framework. We define the decomposition matrix; following [33], we describe a parallel algorithm as an ordered set of operators, moreover we give the definition of complexity of the algorithm depending on the number of such operators; finally, we define the execution matrix describing the mapping of the algorithm on the target computing resource. Section 4 focuses on two metrics characterizing the algorithm performance, such as the scale up factor and the speed up. In Section 5, we analyse the performance of parallel algorithms arising from the same problem decomposition. We derive the Generalized Amdhal's Law and some important upper and lower bounds of the performance metrics. In Section 6, we consider the particular case where the operators of an algorithm have the same execution time (namely, the operators are the usual floating point operations); in other words, we are assuming to get a decomposition at the lowest level of granularity and we derive the standard expressions for the performance metrics. Section 7 introduces the access memory matrix and some useful performance metrics to evaluate specific aspects of the software implementation. In Section 8 conclusions are drawn.

1.1 Related Works and Criticism

The appropriate mapping depends upon both the specification of the algorithm and the underlying architecture. Firstly, it implies a transformation of the algorithm into an equivalent, but in a more appropriate form. Works on the mapping problem can be classified according to the used representation.

Graph based approaches perform transformations on the algorithm and the architecture, both represented as graphs. In this approach the algorithm is modeled in terms of graphs structures and the mapping in terms of graphs partitions [8]. Linear algebra approaches represent the graph and its data dependencies by a matrix, then they transform the graph by performing matrix operations. Language based approaches transform one form of program text into another form, where the target

form textually incorporates information about the architecture [25]. Characteristic based approaches represent the algorithm in terms of a set of characteristics which determines the transformations. Included in this category is the work of [28], where a technique which abstracts a computation in terms of its data dependencies is described. The method is based on a mathematical transformation of the index sets and of the data-dependency vectors associated with the given algorithm. Finally, we underline that there are a lot of scientific groups that are working on similar issue from the years. In particular we mention the Heterogeneous Computing Laboratory at University College in Dublin focusing on efficient use of heterogeneous architectures. They mainly focus the attention on workload distribution, data distribution, communication performance models and optimization of communication operations of parallel or distributed algorithm on the network, by analyzing partitioning workload in proportion to the speed of the processing elements [29, 35].

One common issue of the aforementioned approaches is that very often the model used for the representation and analysis of the algorithm cannot be explicitly employed for deriving the expression of performance metrics of the software. On the contrary, performance analysis is often accomplished with automatic tools on a combination of the algorithm and the parallel architecture on which it is implemented (the so-called parallel system), exploiting automating mappings, automatic translations, re-targeting mappings tracing, auto-tuning tools (such as: the PaRSEC runtime system [11], that provides a portable way to automatically adapt algorithms to new hardware trend). Nevertheless, these approaches ignore dependency among sub problems within the problem decomposition. Instead, our model, through the choice of the computing operators of the algorithm, allows to set a level of abstraction for the algorithm description; each level determines the granularity/detail of the performance analysis, and could be used to better understand the subsequent mapping on the computing architecture. This topic is mainly important to analyse performance of complex algorithm solving real problems.

2 PRELIMINARY CONCEPTS AND DEFINITIONS

We introduce a dependency relationship among the parts of a computational problem, among operators of the algorithm that solves the problem and, finally, among memory accesses of the algorithm. In this way we are able to define the matrices which are the foundations of the mathematical model we are going to introduce.

To this aim we first give some definitions[1].

Definition 1 (Computational Problem)**.** A computational problem $\mathcal{B}_{N_r}$ is the mathematical problem specified by an input/output functional relation $\mapsto$:

$$\mathcal{B}_{N_r} : In_{\mathcal{B}_{N_r}} \mapsto Out_{\mathcal{B}_{N_r}}$$

[1] It is worth to note that these definitions do not claim to be general. Their aim is to establish the mathematical setting on which we will restrict our attention.

where N_r is the input data size and $r \in \mathbb{N}$, between the data and the solution of $\mathcal{B}_{N_r}$.

In the following the computational problem $\mathcal{B}_{N_r}$ is identified by the triple:

$$\mathcal{B}_{N_r} \equiv \left(N_r, In_{\mathcal{B}_{N_r}}, Out_{\mathcal{B}_{N_r}}\right).$$

Definition 2 (Similar Computational Problem)**.** Two computational problems, $\mathcal{B}_{N_r}$ and $\mathcal{B}_{N_q}$, are said similar if they are specified by the same functional relation $\mapsto$ and they only differ in the input/output data size. If $\mathcal{B}_{N_r}$ and $\mathcal{B}_{N_q}$ are similar we write $\mathcal{B}_{N_r} \mathscr{S} \mathcal{B}_{N_q}$.

Dividing a computation into smaller computations, some or all of which may potentially be executed in parallel, is the key step in designing parallel algorithms. These parts often share input, output, or intermediate data, such that the output of one part is the input of another. In our mathematical framework these relationships will be described by the so called decomposition matrix. In order to define this matrix we need to introduce the following algebraic structure

Definition 3 (Dependency Group)**.** Let $(\mathcal{E}, \pi)$ be a group and let $\pi_{\mathcal{E}}$ be a strict partial order relation on $\mathcal{E}$, which is compatible with π. We say that any element of $\mathcal{E}$, let us say A, depends on an element of $\mathcal{E}$, let us say B, if $A \pi_{\mathcal{E}} B$, and we write $A \leftarrow B$. If A and B do not depend on each other we write $A \nleftarrow B$. The group $(\mathcal{E}, \pi)$ equipped with $\pi_{\mathcal{E}}$ is called dependency group and it is denoted as $(\mathcal{E}, \pi, \pi_{\mathcal{E}})$.

Remark 1. Since $\pi_{\mathcal{E}}$ is transitive, from Definition 2 it follows that any two elements of $\mathcal{E}$, let us say A and B, are independent if there is no any relationship between them. In this case we write $A \nleftarrow B$ and $B \nleftarrow A$, or even $A \nleftrightarrow B$.

Now we are able to define the dependency matrix on $(\mathcal{E}, \pi, \pi_{\mathcal{E}})$.

Definition 4 (Dependency Matrix)**.** Given $(\mathcal{E}, \pi, \pi_{\mathcal{E}})$, the matrix $\mathcal{F}$, of size $r_D \cdot c_D$, whose elements $d_{i,j} \in (\mathcal{E}, \pi)$, are such that $\forall i \in [0, r_D - 1]$

$$d_{i,j} \nleftrightarrow d_{i,s}, \quad \forall s, j \in [0, c_D - 1] \tag{1}$$

and $\forall i \in [1, r_D - 1], \quad \exists q \in [0, c_D - 1]$ s.t.

$$d_{i,j} \leftarrow d_{i-1,q}, \quad \forall j \in [0, c_D - 1], \tag{2}$$

while the others elements are set equal to zero, is said the dependency matrix.

Remark 2. Matrix $\mathcal{F}$ is unique (through its construction), up to a permutation of elements on the same row. $c_{\mathcal{F}}$ is said the concurrency degree[2] of $(\mathcal{E}, \pi, \pi_{\mathcal{E}})$ and $r_{\mathcal{F}}$ is the said the dependency degree of $\mathcal{E}$. Concurrency degree measures the intrinsic concurrency among sub-problems of $(\mathcal{E}, \pi, \pi_{\mathcal{E}})$. It is obtained as the number of columns of $\mathcal{F}$.

[2] A similar concept has already been highlighted in [22]

2.1 The Problem Decomposition

Let $S(\mathcal{B}_{N_r})$ denote the solution[3] of $\mathcal{B}_{N_r}$.

Definition 5 (Decomposition of a Computational Problem)**.** Given $\mathcal{B}_{N_r}$, any finite set of computational problems $\{\mathcal{B}_{N_i}\}_{i=0,\ldots,k-1}$, where $k \in \mathbb{N}$, such that $\mathcal{B}_{N_r} \leftarrow \mathcal{B}_{N_i}$, where $N_i < N_r$, and

$$\sum_{i=0}^{k-1} N_i \geq N_r$$

is called a decomposition of $\mathcal{B}_{N_r}$. $\mathcal{B}_{N_i}$ denotes a sub-problem of $\mathcal{B}_{N_r}$. A decomposition of $\mathcal{B}_{N_r}$, which is denoted as

$$D_k(\mathcal{B}_{N_r}) := \{\mathcal{B}_{N_0}, \ldots, \mathcal{B}_{N_{k-1}}\} \tag{3}$$

defines the computational problem

$$D_k(\mathcal{B}_{N_r}) \equiv \left(\sum_{i=0}^{k-1} N_i, In_{\mathcal{B}_{N_r}}, Out_{\mathcal{B}_{N_r}}\right).$$

The set of all the decompositions of $\mathcal{B}_{N_r}$ is denoted as $\mathcal{DB}_{N_r}$.

Definition 6 (Similar Decompositions)**.** Given $\mathcal{B}_{N_r} \mathscr{S} \mathcal{B}_{N_q}$, two decompositions $D_{k_i}(\mathcal{B}_{N_r})$ and $D_{k_j}(\mathcal{B}_{N_q})$ are called similar if

$$k_i = card(D_{k_i}(\mathcal{B}_{N_r})) = card(D_{k_j}(\mathcal{B}_{N_q})) = k_j$$

and

$$\forall \mathcal{B}_{N_s} \in D_{k_i}(\mathcal{B}_{N_r}) \exists! \mathcal{B}_{N_t} \in D_{k_j}(\mathcal{B}_{N_q}) : \mathcal{B}_{N_s} \mathscr{S} \mathcal{B}_{N_t}$$

and we write

$$D_{k_i}(\mathcal{B}_{N_r}) \mathscr{S} D_{k_j}(\mathcal{B}_{N_q}).$$

Remark 3 (Decomposition Matrix)**.** In order to capture interactions among parts (or sub-problems) of $\mathcal{B}_{N_r}$, we use the dependency matrix on $D_k(\mathcal{B}_{N_r})$. More precisely, by using Definition 2 we introduce the group $(D_k(\mathcal{B}_{N_r}), g_{sol})$ where g_{sol} is any application between any two elements $\mathcal{B}_{N_i}$ and $\mathcal{B}_{N_j}$ of $D_k(\mathcal{B}_{N_r})$, equipped with the strict partial order relation $\pi_{D_k(\mathcal{B}_{N_r})}$. Then, we construct the (unique) dependency matrix $\mathcal{F}$ corresponding to the decomposition $D_k(\mathcal{B}_{N_r})$. In the following we denote this matrix as $M_D(D_k(\mathcal{B}_{N_r}))$, or M_{D_k} for simplicity, and we refer to it as the **decomposition matrix**. Given $D_k(\mathcal{B}_{N_r})$, let c_{D_k} denote the number of columns. This is the (unique) concurrency degree of $\mathcal{B}_{N_r}$. Let r_{D_k} denote the row number of rows. This is the (unique) dependency degree of $\mathcal{B}_{N_r}$. Concurrency degree measures the intrinsic concurrency among sub-problems of $\mathcal{B}_{N_r}$.

[3] Here, for the sake of simplicity, we assume that $S(\mathcal{B}_{N_r})$ exists and it is unique.

If there are not empty elements, the problem $\mathcal{B}_{N_r}$ has the highest intrinsic concurrency, hence we give the following

Definition 7 (Perfectly Decomposed Problems)**.** $\mathcal{B}_{N_r}$ is said perfectly decomposed if $\exists D_k(\mathcal{B}_{N_r})$ and M_D such that

- $c_D > 1$,
- $\forall i, j,\ d_{i,j} \neq \emptyset$.

The next step is to take these parts and assign them (i.e., the mapping step) onto the computing machine. In the next section we introduce the computing environment characterized by the set of logical-operational operators/operations that it is able to apply/execute and by the memory system.

2.2 The Computing Architecture

Let $\mathcal{M}_P$ denote the computing architecture equipped with $P \geq 1$ processing elements with specific logical-operational capabilities such as: basic operations (arithmetic, ...), special functions evaluations (sin, cos, ...), solvers (integrals, equations system, non linear equations, ...). These are the computing operators of $\mathcal{M}_P$. In particular, we will use the following characterization of operators of $\mathcal{M}_P$.

Definition 8 (Computing Operators)**.** The operator I^j of $\mathcal{M}_P$ is a correspondence between $\mathbb{R}^s$ and $\mathbb{R}^t$, where $s, t \in \mathbb{N}$ are positive integers.

Given $\mathcal{M}_P$, the set

$$Cop_{\mathcal{M}_P} := \{I^j\}_{j\in[0,q-1]}, \quad q \in \mathbb{N}$$

where operators I^j are taken without repetitions, characterizes logical-operational capabilities of $\mathcal{M}_P$.

Operators, properly organized, provide the solution to $\mathcal{B}_{N_r}$, as stated in the following definition.

Definition 9 (Solvable Problems)**.** $\mathcal{B}_{N_r}$ is solvable in $\mathcal{M}_P$ if

$$\exists D_k(\mathcal{B}_{N_r}) \in \mathcal{DB}_{N_r} : \forall \mathcal{B}_{N_j} \in D_k(\mathcal{B}_{N_r}) \qquad \exists I^j \in Cop_{\mathcal{M}_P} : I^j[\mathcal{B}_{N_j}] = S(\mathcal{B}_{N_j})$$

that is, if there exists any relation

$$\theta : \mathcal{B}_{N_j} \in D_k(\mathcal{B}_{N_r}) \in \mathcal{DB}_{N_r} \longmapsto I^j \in Cop_{\mathcal{M}_P}. \tag{4}$$

In particular, we say that a decomposition is suited for $\mathcal{M}_P$ if θ is a function. From now on, we assume any problem $\mathcal{B}_{N_r}$ to be solvable, and we assume a decomposition $D_k(\mathcal{B}_{N_r}) \in \mathcal{DB}_{N_r}$ suited for $\mathcal{M}_P$ to be fixed[4].

We associate to each $I^i \in Cop_{\mathcal{M}_P}$ in $\mathcal{M}_P$ the parameter t_i, denoting the execution time measured, for instance, in seconds. If $I^i \equiv \varnothing$, we set $t_\varnothing = 0$.

[4] Note that there is no loss of generality.

2.3 Memory Hierarchy and Communications

A computing architecture is not only characterized by the set of operations that is able to apply, but also by the memory system. Indeed, the effective performance of an algorithm relies not only on the processor speed for arithmetic operations but also on the ability of the memory system to feed data to the processor. At the logical level, a memory system, possibly consisting of multiple levels of caches, let us say L, takes in a request for a memory word and returns a block of data containing the requested word after $tmem_l$ nanoseconds. Here, $tmem_l$ consists of memory latency time, measuring the time between the of a read request and the release of its corresponding data, plus the data transfer time. Memory latency time depends on the latency of the memory, which is typically bridged by a hierarchy of successively faster memory that rely on locality of data reference to deliver higher performance. The rate at which data can be moved from the memory to the processor determines the bandwidth of the memory system. It is determined by the memory bus bandwidth as well as by the memory unit.

So, we consider a computing machine $\mathcal{M}_P$ such that

- its memory has $L \geq 2$ levels,
- for each level l, where $l \in [1, L]$, nd_l denotes the bandwidth, i.e. the rate for transferring (read/write) data of the same type. It is $nd_l \geq 1$,
- memory access time is
$$tmem_l := t_l^{acc} + t_l^{trans}$$
where t_l^{acc} measures the memory latency time while t_l^{trans} measures the transfer time. Moreover, let $tcalc$ be the execution time of one floating point operation. We assume that
$$tcalc < tmem_1 \leq tmem_2 \leq \ldots \leq tmem_L$$

and

$$tmem_l = \alpha_l^{mem} \cdot tmem, \quad \forall l \in [1, L], \quad \alpha_l \in \Re - \{\infty\} \tag{5}$$

where $tmem$ denotes the execution time needed for moving a memory word.

Remark 4. In case of latency bound algorithms (i.e., t_l^{acc} prevails over t_l^{trans}) or bandwidth bound algorithms (i.e., t_l^{trans} prevails over t_l^{acc}) the model could be properly refined by specifying $tmem_l$.

Communication means moving data, either between levels of a memory hierarchy or between processors of the reference machine. Hence, this mathematical framework includes the communication level within the memory accesses, and the last (that is the slowest) memory level (L^{th}) refers to it. Let $tcom := tmem_L$ denote the unitary communication time, we assume that $tcom \gg tmem_i, i \in [1, L-1]$.

Such computing machine is denoted as $\mathcal{M}_{P,L,nd_L}$ or simply as $\mathcal{M}_P$ if there is no ambiguity.

3 THE ALGORITHM

In literature, an algorithm is any procedure consisting of a finite number of unambiguous rules that specify a finite sequence of operations bringing to a solution of a problem or of a specific class of problems [24]. Analogously, we define an algorithm as a finite set of operators solving $\mathcal{B}_{N_r}$.

Definition 10 (Algorithm)**.** Given $D_k(\mathcal{B}_{N_r})$, an algorithm solving $\mathcal{B}_{N_r}$, indicated as

$$A_{D_k(\mathcal{B}_{N_r}),\mathcal{M}_P} = \{I^{i_0}, I^{i_1}, \ldots, I^{i_k}\}$$

is a sequence of elements (not necessarily distinct) of $Cop_{\mathcal{M}_P}$, such that[5]

$$I^{i_k} \circ I^{i_{k-1}} \circ \ldots \circ I^{i_0}[\mathcal{B}_{N_r}] = S(\mathcal{B}_{N_r})$$

where $j \in [0, card(Cop_{\mathcal{M}_P}) - 1]$, and such that there is a bijective correspondence

$$\gamma : \mathcal{B}_{N_\nu} \in D_k(\mathcal{B}_{N_r}) \in \mathcal{D}\mathcal{B}_{N_r} \longleftrightarrow I^{i_j} \in A_{D_k(\mathcal{B}_{N_r}),\mathcal{M}_P}. \tag{6}$$

Every ordered subset of $A_{D_k(\mathcal{B}_{N_r}),\mathcal{M}_P}$ is a sub-algorithm of $A_{D_k(\mathcal{B}_{N_r}),\mathcal{M}_P}$.

For simplicity of notations and when there is no ambiguity, we indicate algorithms briefly as $A_{k,P}$.

Definition 11 (Equal Algorithms)**.** Two algorithms

$$A^i_{k,P} = \{I^{i_0}, I^{i_1}, \ldots, I^{i_k}\}, \quad A^j_{k,P} = \{I^{j_0}, I^{j_1}, \ldots, I^{j_k}\}$$

are said equal if $\forall s \in [0, k], I^{i_s} \equiv I^{j_s}$.

We note that in this mathematical framework, *two equal algorithms have the same cardinality.*

Definition 12 (Granularity Set of an Algorithm)**.** Given $A_{k,P}$, the subset $\mathcal{G}(A_{k,P})$ of $A_{k,P}$ made of distinct operators of $A_{k,P}$ defines the granularity set of $A_{k,P}$. Two algorithms

$$A^i_{k,P} = \{I^{i_0}, I^{i_1}, \ldots, I^{i_k}\}, \quad A^j_{k,P} = \{I^{j_0}, I^{j_1}, \ldots, I^{j_k}\}$$

have the same granularity if $\mathcal{G}(A^i_{k,P}) \equiv \mathcal{G}(A^j_{k,P})$.

Let $AL_{\mathcal{B}_{N_r}}$ (or simply AL) be the set of algorithms that solve $\mathcal{B}_{N_r}$, obtained by varying $\mathcal{M}_P$, the number of processing units P and $D_k(\mathcal{B}_{N_r}) \in \mathcal{D}\mathcal{B}_{N_r}$. Even if one can easily formulate infinite variations of an algorithm that do the same thing, for simplicity in the following we assume AL to be finite.

[5] In the following we use the symbol $\circ$ to denote correspondence composition.

Definition 13 (The Quotient Set $\frac{AL}{\varrho}$)**.** Let

$$\varphi : A_{k,P} \in AL \longrightarrow D_k(\mathcal{B}_{N_r}) \in \mathcal{DB}_{N_r} \tag{7}$$

be the surjective correspondence which induces on AL an equivalence relationship ϱ of AL in itself, such that

$$\varrho(A_{k,P}) = \{\widetilde{A_{k,P}} \in AL : \varphi(\widetilde{A_{k,P}}) = \varphi(A_{k,P})\}. \tag{8}$$

The set $\varrho(A_{k,P})$ consists of algorithms of AL associated with the same decomposition $D_k(\mathcal{B}_{N_r}) \in \mathcal{DB}_{N_r}$. ϱ induces the quotient set $\frac{AL}{\varrho}$, whose elements are disjoints and finite subsets of AL determined by ϱ, that is they are equivalence classes under ϱ.

In the following we assume $A_{k,P}$ to represent its equivalence class in AL.

Definition 14 (Complexity)**.** The cardinality of $A_{k,P}$, denoted as $C(A_{k,P})$, is said complexity of $A_{k,P}$. It is $C(A_{k,P}) := card(A_{k,P}) = k$.

Remark 5. $C(A_{k,P}) = k$ equals to the number of non empty elements of M_{D_k}, i.e. the decomposition matrix defined on $D_k(\mathcal{B}_{N_r})$. By virtue of the bijective correspondence γ in (6), it holds that

$$card(A_{k,P}) = card(D_k(\mathcal{B}_{N_r})) = k, \quad \forall A_{k,P} \in \varrho(A_{k,P}). \tag{9}$$

Each algorithm belonging to the same equivalence class according to ϱ has the same complexity. An integer (the complexity) is therefore associated with each element $\varrho(A_{k,P})$ of quotient set $\frac{AL}{\varrho}$ which induces an ordering relation between the equivalence classes in $\frac{AL}{\varrho}$: therefore there is a minimum complexity for algorithms that solve the problem $\mathcal{B}_{N_r}$.

Remark 6 (Similar Algorithms)**.** Given $\mathcal{B}_{N_r}\mathscr{S}\mathcal{B}_{N_q}$ and their relative similar decompositions $D'_k(\mathcal{B}_{N_r})\mathscr{S}D''_k(\mathcal{B}_{N_q})$ (see Definition 6), algorithms belonging to $\varrho(A_{k,P}) = \varphi^{-1}(D'_k(\mathcal{B}_{N_r}))$ (see (7)) are similar to algorithms belonging to $\varrho(A_{k,P}) = \varphi^{-1}(D''_k(\mathcal{B}_{N_q}))$. From Definition 6 and 14 and (9), it follows that

$$A_{k,P}\mathscr{S}A_{k,P} \Longrightarrow C(A_{k,P}) = C(A_{k,P}) = k,$$

that is, *similar algorithms have the same complexity.*

Remark 7. As we can associate $I^{i_k} \in A_{k,P}$ to each subproblem according to γ, then the operators of $A_{k,P}$ inherit the dependencies existing between subproblems of $\mathcal{B}_{N_r}$, but they do not inherit independency, because for instance, two operators may depend on the availability of computing units of $\mathcal{M}_P$ during their execution [33].

Remark 8 (Execution Matrix)**.** According to Definition 3, we introduce the group $\left(\mathcal{P}\left(A_{k,P}\right), \circ, \pi_{A_{k,P}}\right)$ where $\mathcal{P}\left(A_{k,P}\right)$ is the set of all the sub-algorithms of $A_{k,P}$, and

$\pi_{A_{k,P}}$ is the strict partial order relation between any two elements of $\mathcal{P}(A_{k,P})$ that guarantees that two elements cannot be performed in any arbitrary order and simultaneously[6]. We construct matrix $\mathcal{F}$ of order $r_E \cdot c_E$, where $c_E = P$[7] as a dependency matrix (see Definition 4). The number of columns of this matrix will represent the maximum number of sub-algorithms that can be performed simultaneously on $\mathcal{M}_P$. In the following, we denote this matrix as **execution matrix** and we refer to it by using the symbol $M_E(A_{k,P}) = (e_{i,j})$ or simply $M_{E_{k,P}}$ if there is no ambiguity. Matrix $M_{E_{k,P}}$ is unique up to a permutation of elements on the same row. This matrix can be placed in analogy with the execution graphs (see [7, 10, 12, 30]) that are often used to describe the sequence of steps of an algorithm on a given machine for a particular input or a particular configuration.

Remark 9. As it is $card(A_{k,P}) = card(D_k(\mathcal{B}_{N_r}))$, then M_{D_k} and $M_{E_{k,P}}$ have the same number of non empty elements (k), whichever is $P \geq 1$. If $c_E = P = c_{D_k}$, there exists $A_{k,P}$ whose matrix $M_{E_{k,P}}$ has exactly the same structure of the matrix M_{D_k}.

Definition 15. $A_{k,P}$ is said perfectly parallel if: $c_E > 1$ and $\forall i, j : e_{i,j} \neq \emptyset$. $A_{k,P}$ is said sequential if: $c_E = 1$ and $\nexists j > 1 : e_{i,j} \neq \emptyset$. $A_{k,P}$ is said (simply) parallel if: $c_E > 1$ and $\exists i, j : e_{i,j} = \emptyset$. Moreover, every row of matrix $M_{E_{k,P}}$ such that $\exists e_{i,j} \neq \emptyset$, where $j > 1$, is a parallel sub-algorithm of $A_{k,P}$. Every row of matrix $M_{E_{k,P}}$ such that $\exists! e_{i,j} \neq \emptyset$ is a sequential sub-algorithm of $A_{k,P}$.

Remark 10. Observe that the concurrency degree of $\mathcal{B}_{N_r}$ in a given decomposition provides an upper limit to the maximum number of independent sub-algorithms executable simultaneously on the machine. The dependency degree provides a lower limit to the execution time of the algorithm.

Finally, from correspondence γ (see (6)), we say that $\mathcal{B}_{N_r}$ is solvable in $\mathcal{M}_P \Leftrightarrow \exists D_k(\mathcal{B}_{N_r}) \in \mathcal{D}\mathcal{B}_{N_r} : \exists A_{k,P}$ that solves $\mathcal{B}_{N_r}$.

Theorem 1. If $\mathcal{B}_{N_r}$ is perfectly decomposed according to D_k, $\exists \mathcal{M}_P$, where $P > 1$, such that $\exists A_{k,P}$ perfectly parallel that solves $\mathcal{B}_{N_r}$.

Proof. If $\mathcal{B}_{N_r}$ is perfectly decomposed then the matrix M_{D_k} has not empty elements and has order greater than 1. Since $card(A_{k,P}) = card(D_k(\mathcal{B}_{N_r})) = k$, there exists $A_{k,P}$ with execution matrix $M_{E_{k,P}}$ of order $r_E \cdot c_E$, with only non zero elements,

[6] The condition that two elements cannot be performed in any arbitrary order induces the inheritance of dependencies between decomposition subproblems and algorithm operators, while the condition that two elements cannot be performed simultaneously – relating to availability of resources – adds possible reasons for dependency between operators, which depend on the machine on which algorithm A is intended to run [33].

[7] In general $c_E \leq P$, but we can exclude cases where dependencies existing between subproblems do not allow to use all the computing units available, i.e., in which $c_E < P$, because they can be easily taken back to the case where $c_E = P$.

such that $r_E = r_{D_k}$ and $c_E = P = c_{D_k}$ or[8] $r_E = n \cdot r_{D_k}$ and $c_E = P = c_{D_k}/n$ with the integer n is such that $n < c_{D_k}$ and $c_{D_k} \bmod n = 0$. In conclusion, $M_{E_{k,P}}$ has $c_E = P > 1$ columns, and no rows have an empty element; so $A_{k,P}$ is perfectly parallel. □

4 PERFORMANCE METRICS

In this section we employ the mathematical settings we introduced in Section 2, in order to define two quantities to measure the performance of an algorithm: the scale up and the speed up.

Let us consider two decompositions $D_{k_i}(\mathcal{B}_N)$ and $D_{k_j}(\mathcal{B}_N)$ in $\mathcal{DB}_N$. Let us consider $A_{k_i,P}$ and $A_{k_j,P}$ representing their equivalence class in AL. We introduce the following quantity:

Definition 16 (Scale Up Factor)**.** If $A_{k_i,P}$ and $A_{k_j,P}$ have the same granularity set (see Definition 12), the ratio

$$Sc_{up}(A_{k_i,P}, A_{k_j,P}) := \frac{k_i}{k_j} \tag{10}$$

is said scale up factor of $\varrho(A_{k_j,P})$ measured with respect to $\varrho(A_{k_i,P})$.

Note that by using Definition 14, we get

$$Sc_{up}(A_{k_i,P}, A_{k_j,P}) = \frac{C(A_{k_i,P})}{C(A_{k_j,P})}. \tag{11}$$

Next proposition quantifies the scale up when we solve the same problem with an algorithm that is the concatenation of several algorithms which are similar to the first one, with polynomial complexity of degree d.

Proposition 1. Given $\mathcal{B}_{N_r}$, $D_k(\mathcal{B}_{N_r})$ and $D_{k'}(\mathcal{B}_{N_r}) = \{D_{k'_i}(\mathcal{B}_{N_q})\}_{i=1,\mu}$ where

- $N_q = N_r/\mu$ with $\mu \in N$, $\mu \leq N_r$, and $N_r \bmod \mu = 0$,
- $\mathcal{B}_{N_q} \mathscr{S} \mathcal{B}_{N_r}$,
- $D_k \mathscr{S} D_{k'_i} \mathscr{S} D_{k'_j}$, $\forall i \neq j$.

Consider $A_{k,P} \in \varphi^{-1}(D_k(\mathcal{B}_{N_r}))$ and $A_{k'_i,P} \in \varphi^{-1}(D_{k'_i}(\mathcal{B}_{N_q}))$ and assume that $C(A_{k,P}) = k = \mathcal{P}^d(N_r)$ and $C(A_{k'_i,P}) = k'_i = \mathcal{P}^d(N_q)$ where

$$\mathcal{P}^d(x) = a_d x^d + a_{d-1} x^{d-1} + \ldots + a_0, \quad a_d \neq 0 \in \Pi_d, x \in \Re,$$

[8] If the concurrency degree c_{D_k} is so great that we cannot imagine a real machine with so many units, we can always use a number of computing units $P = c_{D_k}/n$ with $c_{D_k} \bmod (n) = 0$. This will mean that the execution matrix of $A_{k,P}$ will have n times more rows and n times less columns than the dependency matrix.

then $Sc_{up}(A_{k,P}, A_{k',P}) = \xi(N_r, \mu) \cdot \mu^{d-1}$ where

$$\xi(N_r, \mu) := \frac{a_d + \frac{a_{d-1}}{N_r} + \ldots + \frac{a_0}{N_r^d}}{a_d + a_{d-1}\frac{\mu}{N_r} + \ldots + a_0\frac{\mu^d}{N_r^d}}. \tag{12}$$

Proof. We have that

$$C(A_{k',P}) = \sum_{i=1}^{\mu} C(A_{k'_i,P}) = \mu \cdot \mathcal{P}^d(N_q), \tag{13}$$

then from the (11) it follows that

$$Sc_{up}(A_{k,P}, A_{k',P}) = \frac{C(A_{k,P})}{C(A_{k',P})} = \frac{\mathcal{P}^d(N_r)}{\mu \cdot \mathcal{P}^d(N_q)}, \tag{14}$$

that is

$$Sc_{up}(A_{k,P}, A_{k',P}) = \frac{a_d N_r^d + a_{d-1} N_r^{d-1} + \ldots + a_0}{\mu \cdot \left(a_d N_q^d + a_{d-1} N_q^{d-1} + \ldots + a_0\right)}. \tag{15}$$

Since $N_q = N_r/\mu$, then it is

$$Sc_{up}(A_{k,P}, A_{k',P}) = \frac{a_d (\mu N_q)^d + a_{d-1} (\mu N_q)^{d-1} + \ldots + a_0}{\mu \cdot \left(a_d N_q^d + a_{d-1} N_q^{d-1} + \ldots + a_0\right)}, \tag{16}$$

then thesis follows from the (12). □

It comes out the following result:

Corollary 1. If N_r is fixed, and $\mu \simeq N_r$, it is $\xi(N_r, \mu) = const$ where $const \in (0, 1]$ and $Sc_{up}(A_{k,P}, A_{k',P}) \leq N_r^{d-1}$. If μ is fixed, it is $\lim_{N_r \to \infty} \xi(N_r, \mu) = const$ where $const \in (0, 1]$ and $\lim_{N_r \to \infty} Sc_{up}(A_{k,P}, A_{k',P}) \leq \mu^{d-1}$. If $a_i = 0$, $\forall i < d$, then $\xi(N_r, \mu) = 1$ and $S_{up}(A_{k,P}, A_{k',P}) = \mu^{d-1}$, $\forall \mu$.

We observe that in the following, when we need to refer to the execution time of computing operators of $A_{k,P}$, we will use the notation $\beta^{calc}_{\ldots, M_{E_{k,P}}}$ for the parameters highlighting the execution matrix $M_{E_{k,P}}$ and characterizing the mapping of the algorithm on the machine $\mathcal{M}_P$. We assume that

$$\forall I^{i_j} \in Cop_{\mathcal{M}_P}, \quad t_{i_j} = \beta^{calc}_{i_j, M_{E_{k,1}}} \cdot tcalc, \beta^{calc}_{i_j, M_{E_{k,1}}} \in \Re, \quad \beta^{calc}_{i_j, M_{E_{k,1}}} \geq 1. \tag{17}$$

Definition 17 (Row Execution Time)**.** The quantity

$$T_r(A_{k,P}) := \max_{j \in [0, c_E - 1]} t_{r_j} \tag{18}$$

is said execution time of the row r of $M_{E_{k,P}}$ (which is a sub-algorithm of $A_{k,P}$).

Remark 11. Let $\beta^{calc}_{r,M_{E_{k,P}}} := \max_{j\in[0,c_E-1]} \beta^{calc}_{r_j,M_{E_{k,P}}}$, then

$$T_r(A_{k,P}) = \max_{j\in[0,c_E-1]} \beta^{calc}_{r_j,M_{E_{k,P}}} \cdot tcalc = \beta^{calc}_{r,M_{E_{k,P}}} \cdot tcalc.$$

Note that $\beta^{calc}_{i_j,M_{E_{k,1}}} \geq 1$, then $\beta^{calc}_{r,M_{E_{k,1}}} \geq 1$.

Definition 18 (Execution Time)**.** The quantity

$$T(A_{k,P}) := \sum_{r=0}^{r_E-1} T_r(A_{k,P}) \tag{19}$$

is said execution time of $A_{k,P}$.

Remark 12. Let $\beta^{calc}_{M_{E_{k,P}}} := \sum_{r=0}^{r_E-1} \beta^{calc}_{r,M_{E_{k,P}}}$, then $\beta^{calc}_{M_{E_{k,P}}} \geq r_E$.

$$T(A_{k,P}) = \beta^{calc}_{M_{E_{k,P}}} \cdot tcalc. \tag{20}$$

Remark 13. Let

$$\beta^{calc}_{sum,M_{E_{k,P}}} := \sum_{i=0}^{r_E-1} \sum_{j=0}^{c_E-1} \beta^{calc}_{i_j,M_{E_{k,P}}}. \tag{21}$$

Then, if $P = 1$, then $\beta^{calc}_{M_{E_{k,P}}} := \beta^{calc}_{sum,M_{E_{k,P}}}$.

Remark 14. Let

- $r_{seq} \leq r_E$ denote the number of rows of $M_{E_{k,P}}$ with only one non-empty element (sequential sub-algorithms of $A_{k,P}$);
- $r_{par} = r_E - r_{seq}$, with $r_{par} \leq r_E$, denote the number of rows of $M_{E_{k,P}}$ with more than one non empty element.

From the sequence $i = 0, \ldots, r_E - 1$, numbering the r_E rows of $M_{E_{k,P}}$, two sub-sequences of indices originate $\{i_q\}_{q\in[0,r_{seq}-1]}$, and $\{i_r\}_{r\in[0,r_{par}-1]}$, and the following definition follows.

Definition 19 (Parallel Execution Time)**.** The quantity

$$T_{par}(A_{k,P}) := \sum_{r=0}^{r_{par}-1} T_{i_r}(A_{k,P}) \tag{22}$$

is said parallel execution time of $A_{k,P}$.

Definition 20 (Sequential Execution Time)**.** The quantity

$$T_{seq}(A_{k,P}) := \sum_{q=0}^{r_{seq}-1} T_{i_q}(A_{k,P}) \tag{23}$$

is said sequential execution time of $A_{k,P}$.

The (19) can be written as

$$T(A_{k,P}) = T_{seq}(A_{k,P}) + T_{par}(A_{k,P}). \tag{24}$$

This states that, by looking at matrix $M_{E_{k,P}}$, the model expresses the size of the parallel and the sequential parts composing the execution time $A_{k,P}$.

Let

$$R^{calc}(A_{k,P}) := \frac{\beta^{calc}_{M_{E_{k,P}}}}{r_E}. \tag{25}$$

R^{calc} is the parameter of the algorithm $A_{k,P}$ depending on the most computationally intensive sub-algorithms of A.

It holds

$$T(A_{k,P}) = R^{calc}(A_{k,P}) \cdot r_E \cdot tcalc = \sum_{r=0}^{r_E-1} \beta^{calc}_{r,M_{E_{k,P}}} \cdot tcalc. \tag{26}$$

Remark 15. If $P = 1$, since $r_E = C(A_{k,1}) = k$ from (25), it is

$$R^{calc}(A_{k,1}) := \frac{\beta^{calc}_{all,M_{E_{k,P}}}}{k}. \tag{27}$$

Corollary 2. From the (25) it follows

$$T(A_{k,1}) = k \cdot R^{calc}(A_{k,1}) \cdot tcalc, \tag{28}$$

$$T(A_{k,P}) \geq r_D \cdot R^{calc}(A_{k,P}) tcalc \tag{29}$$

and it assumes its minimum value when $r_E = r_D$.

$$T(A_{k,P}) = (r_{seq} + r_{par}) \cdot R^{calc}(A_{k,P}) \cdot tcalc. \tag{30}$$

Definition 21 (Speed Up in $\frac{AL}{\rho}$)**.** Given

- $\mathcal{B}_{N_r}$,
- $A_{k,P} \in \varphi^{-1}(D_k(\mathcal{B}_{N_r}))$ where $P > 1$,
- two different decompositions $D_k(\mathcal{B}_{N_r})$ and $D_{k'}(\mathcal{B}_{N_r})$,
- $A_{k',1} \in \varphi^{-1}(D_{k'}(\mathcal{B}_{N_r}))$

where $\mathcal{M}_1$ and $\mathcal{M}_P$ differ only in the number of processing elements, if $\mathcal{G}(A_{k,P}) = \mathcal{G}(A_{k',P})$, then the speed up of $A_{k,P}$ with respect to $A_{k',1}$ is

$$Sp(A_{k,P}, A_{k',1}) := Sc_{up}(A_{k,P}, A_{k',1}) \cdot \frac{T(A_{k,1})}{T(A_{k,P})} = \frac{k'}{k} \cdot \frac{\beta^{calc}_{sum,M_E(A_{k,P})}}{\beta^{calc}_{M_E(A_{k,P})}}. \tag{31}$$

Remark 16 (Ideal Speed Up)**.** Since it is always[9]

$$\beta^{calc}_{sum,M_E(A_{k,P})} \leq P \cdot \beta^{calc}_{M_E(A_{k,P})},$$

then it holds that

$$Sp(A_{k,P}, A_{k',1}) \leq Sc_{up}(A_{k,P}, A_{k',1}) \cdot P. \tag{32}$$

Definition 22 (Speed Up in $\rho(A_{k,P})$)**.** The speed up of $A_{k,P}$ with respect to $A_{k,1}$ is

$$Sp(A_{k,P}) = \frac{T(A_{k,1})}{T(A_{k,P})} = \frac{\beta^{calc}_{sum,M_E(A_{k,P})}}{\beta^{calc}_{M_E(A_{k,P})}}. \tag{33}$$

Example 1. Preliminary results on speed up and scale up validating this approach appear in [27, 3, 14]. In [27], the authors addressed the development of a modular implementation of MGRIT (MultiGrid-In-Time), a parallel iterative algorithm to solve linear and nonlinear systems that arise from the discretization of evolutionary models with a parallel in time approach in the context of the PETSc (the Portable, Extensible Toolkit for Scientific computing) library. The algorithm speed up has been analyzed a priori to provide the best number of processing elements and grid levels needed to address the scaling of MGRIT. In [3, 14], the performance analysis carried out by the authors using the scale up factor suggests the introduction of a highly scalable problem decomposition.

5 ALGORITHMS IN THE SAME EQUIVALENCE CLASS

We consider algorithms that are in the same equivalence class, i.e. those corresponding to the same decomposition of the problem

Theorem 2. $\forall \mathcal{B}_{N_r}$ perfectly decomposed according to the decomposition $D_k(\mathcal{B}_{N_r})$, and $\forall A_{k,P}$ perfectly parallel algorithm that solves it on $\mathcal{M}_P$ with $P > 1$, if

$$Cop_{\mathcal{M}_1} \equiv Cop_{\mathcal{M}_p},$$

it follows that:

$$T(A_{k,P}) = \frac{T(A_{k,1})}{P} \cdot \frac{R^{calc}(A_{k,P})}{R^{calc}(A_{k,1})}. \tag{34}$$

Proof. If $A_{k,P}$ is perfectly parallel, then $M_{E_{k,P}}$ has no empty elements so

$$r_E = \frac{k}{c_E} = \frac{k}{P}.$$

[9] $\beta^{calc}_{M_E(A_{k,P})}$ is the sum of the maximum operator time on each row, so $\beta^{calc}_{sum,M_E(A_{k,P})}$ can be equal to $P \cdot \beta^{calc}_{M_E(A_{k,P})}$ only if the operators have all the same time.

Therefore, from the (26) and (28), it is

$$T(A_{k,P}) = r_E \cdot R^{calc}(A_{k,P}) \cdot tcalc,$$
$$= \frac{k}{c_E} \cdot R^{calc}(A_{k,P}) \cdot tcalc = \frac{T(A_{k,1})}{P} \cdot \frac{R^{calc}(A_{k,P})}{R^{calc}(A_{k,1})}. \quad (35)$$

□

Theorem 3. For all the matrices $M_{E_{k,P}}$ of algorithms in $\varrho(A_{k,P})$, it holds

$$c_E \leq c_{D_k} \quad (36)$$

and

$$r_E \geq r_{D_k}. \quad (37)$$

Moreover, let us consider $A^i_{k,P}$ and $A^j_{k,P}$ two algorithms belonging to $\varrho(A_{k,P})$, and their matrices $M^i_{E_{k,P}}$ and $M^j_{E_{k,P}}$. We have:

- $c^i_E < c^j_E \Rightarrow r^i_E \geq r^j_E$;
- $c^i_E > c^j_E \Rightarrow r^i_E \leq r^j_E$.

Proof. From inheritance on $A_{k,P}$ of dependencies defined on $D_k(\mathcal{B}_{N_r})$, it is not possible that $c_E > c_D$, therefore $c_E \leq c_{D_k}$. Then there is at least one row of M_{D_k} with c_{D_k} non-empty elements. Let d be the difference between c_{D_k} and c_E. Therefore, since M_{D_k} and M_E have the same number of non-empty elements, it is $r_E \geq r_D + \lceil (d/c_{E\cdot}) \rceil$.

Similarly, it can be proved that if $c^i_E < c^j_E$ then $r^i_E \geq r^j_E$, and if $c^i_E > c^j_E$ then $r^i_E \leq r^j_E$. □

Remark 17. The minimum execution time is proportional to the dependency degree of $\mathcal{B}_{N_r}$, that is when the number of computing units is equal to the concurrency degree of $\mathcal{B}_{N_r}$.

We now define a subset of the equivalence class of $\varrho(A_{k,P})$. Let $\simeq$ be the equivalence relation identifying two algorithms with the same P. Then

$$\hat{\varrho}(A_{k,P}) := \varrho(A_{k,P}) / \simeq \quad (38)$$

i.e. consisting of the representatives of the equivalence classes of $\simeq$[10].

Let us now consider matrices $M_{E_{k,P}}$ associated to algorithms belonging to $\hat{\varrho}(A_{k,P})$, varying P.

The following result defines the speed up of a parallel algorithm with respect to the sequential algorithm belonging to its class.

[10] For example, we can take the algorithm in $\hat{\varrho}(A_{k,P})$, $P \geq 1$, whose execution matrix has the fewest number of rows.

Theorem 4. Consider $A_{k,1} \overset{\varrho}{\equiv} A_{k,P}$ with

$$M_{E_1}, \text{ of order } N_1^E = r_{E_1} \cdot 1 \text{ and } M_{E_P} \text{ of order } N_P^E = r_{E_P} \cdot P.$$

It holds

$$Sp(A_{k,P}) = \frac{\beta^{calc}_{sum,M_{E_{k,1}}}}{r_{E_P} \cdot R^{calc}(A_{k,P})}. \tag{39}$$

Proof. From the (26), (27) and (33), it follows

$$\begin{aligned} Sp(A_{k,P}) &= \frac{r_{E_1} \cdot R^{calc}(A_{k,1}) \cdot tcalc}{r_{E_P} \cdot R^{calc}(A_{k,P}) \cdot tcalc} = \frac{C(A_{k,P})}{r_{E_P}} \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})} \\ &= \frac{\beta^{calc}_{sum,M_{E_{k,1}}}}{r_{E_P} \cdot R^{calc}(A_{k,P})}. \end{aligned} \tag{40}$$

□

Corollary 3. Since $(r_{E_P} \cdot c_{E_P}) \geq C(A_{k,P})$, from the (40) it follows that

$$Sp(A_{k,P}) \leq c_{E_P} \cdot \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})} = P \cdot \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}.$$

Definition 23 (Ideal Speed Up in $\hat{\varrho}(A_{k,P})$)**.** We let

$$Sp_{Ideal}(A_{k,P}) = P \cdot \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})} \tag{41}$$

be the ideal speed up.

Let r_{par_i} denote the number of rows having $i > 1$ not empty elements, and $r_{par_1} = r_{seq}$, then it is

$$r_{E_P} = \sum_{i=1}^{P} r_{par_i}.$$

Definition 24 (Total Time of A with i Non Empty Elements)**.** Let T_{j_i} the time of a row with $i \geq 1$ not empty elements. The quantity

$$T_{par_i}(A_{k,P}) = \sum_{j=0}^{r_{par_i}-1} T_{i_j} \tag{42}$$

is the execution time of the part of A with i non empty elements on each row.

Remark 18. It holds that $r_{par} = r_{E_P} - r_{seq} = \sum_{i=2}^{P} r_{par_i}$ then $T_{par_1}(A_{k,P}) = T_{seq}(A_{k,P})$.

Next result shows how the generalized Amdhal's Law can be derived by using the rows of the execution matrix $M_{E_{k,P}}$ having at least one non empty element.

Theorem 5 (Generalized Amdhal's Law)**.** It is

$$Sp(A_{k,P}) = \frac{\frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{\sum_{i=1}^{P} \alpha_i} \tag{43}$$

where

$$\alpha_i = \frac{r_{par_i}}{C(A_{k,P})}.$$

Proof. From (40) it is

$$Sp(A_{k,P}) = \frac{\frac{C(A_{k,P}) \cdot R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{r_{seq} + \sum_{i=2}^{P} r_{par_i}}. \tag{44}$$

By dividing for $C(A_{k,P})$ it follows that

$$Sp(A_{k,P}) = \frac{\frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{\frac{r_{seq}}{C(A_{k,P})} + \sum_{i=2}^{P} \frac{r_{par_i}}{C(A_{k,P})}}, \tag{45}$$

that is

$$Sp(A_{k,P}) = \frac{\frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{\alpha_1 + \sum_{i=2}^{P} \alpha_i}. \tag{46}$$

□

Then, the Amdhal's Law [1] comes out as a particular case of the previous theorem.

Corollary 4 (Amdhal's Law)**.** If we assume that $M_{E_{k,1}}$ only has rows with 1 element or P elements, we have

$$Sp(A_{k,P}) = \frac{\frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{\alpha + \frac{1-\alpha}{P}} \tag{47}$$

where

$$\alpha := \frac{r_{seq}}{C(A_{k,P})}.$$

Proof. From (43) it follows that

$$Sp(A_{k,P}) = \frac{\frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{\alpha_1 + \sum_{i=2}^{P} \alpha_i} \tag{48}$$

where

$$\alpha_i := \frac{r_{par_i}}{C(A_{k,P})}$$

and

$$\frac{r_{par}}{C(A_{k,P})} = \sum_{i=2}^{P} \alpha_i.$$

If the rows with more than one non empty element have P elements, it is

$$r_{par} = \frac{C(A_{k,P}) - r_{seq}}{P},$$

therefore, if we let $\alpha_1 = \alpha = \frac{r_{seq}}{C(A_{k,P})}$, we get

$$Sp(A_{k,P}) = \frac{\frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{\frac{r_{seq}}{C(A)} + \frac{r_{par}}{C(A_{k,P})}} = \frac{\frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{\frac{r_{seq}}{C(A_{k,P})} + \frac{C(A_{k,P}) - r_{seq}}{C(A_{k,P}) \cdot P}} = \frac{\frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{\alpha + \frac{1-\alpha}{P}}. \tag{49}$$

□

Let Q denote the cost of $A_{k,P}$. The cost is defined as the product of the execution time and the number of processors utilized [19]. In this mathematical settings it holds that the cost Q can be written as

$$Q(A_{k,P}) = c_E \cdot r_E \cdot R^{calc}(A_{k,P}) \cdot tcalc. \tag{50}$$

If $c_E = 1$, from the (28) it holds

$$\begin{aligned} Q(A_{k,1}) &= r_E \cdot R^{calc}(A_{k,P}) \cdot tcalc = T(A_{k,1}) = C(A_{k,P}) \cdot R^{calc}(A_{k,1}) \cdot tcalc \\ &= \beta^{calc}_{sum,M_{E_{k,1}}} \cdot tcalc. \end{aligned} \tag{51}$$

The overhead of $A_{k,P}$ is the total time spent by all the processing elements over and above that spent in useful computation.

Definition 25 (Algorithm Overhead)**.** The quantity

$$Oh(A_{k,P}) := Q(A_{k,P}) - Q(A_{k,1}) = \left(c_E \cdot \beta^{calc}_{M_{E_{k,P}}} - \beta^{calc}_{sum,M_{E_{k,1}}} \right) \cdot tcalc \tag{52}$$

is said overhead of $A_{k,P}$.

Theorem 6. It holds

$$C(A_{k,P}) \cdot (R^{calc}(A_{k,P}) - R^{calc}(A_{k,1})) \cdot tcalc \begin{cases} = 0, & \text{if } R^{calc}(A_{k,P}) = R^{calc}(A_{k,1}), \\ > 0, & \text{otherwise.} \end{cases} \tag{53}$$

Proof. It holds

$$\begin{aligned} Q(A_{k,P}) &\geq card(A_{k,P}) \cdot R^{calc}(A_{k,P}) \cdot tcalc \\ &= C(A_{k,P}) \cdot R^{calc}(A_{k,P}) \cdot tcalc = k \cdot R^{calc}(A_{k,P}) \cdot tcalc. \end{aligned} \tag{54}$$

Moreover,

$$Q(A_{k,1}) = C(A_{k,1}) \cdot R^{calc}(A_{k,1}) \cdot tcalc = k \cdot R^{calc}(A_{k,1}) \cdot tcalc,$$

therefore it follows from (52)

$$Oh(A_{k,P}) \geq \big(k \cdot (R^{calc}(A_{k,P}) - R^{calc}(A_{k,1}))\big) \cdot tcalc$$

and (53) follows. □

Definition 26 (Ideal Overhead in $\hat{\varrho}(A_{k,P})$)**.** From the (53) it follows

$$Oh_{Ideal}(A_{k,P}) = \big(k \cdot (R^{calc}(A_{k,P}) - R^{calc}(A_{k,1})\big) \cdot tcalc. \tag{55}$$

Let $Ef(A_{k,P}) := \frac{Sp(A_{k,P})}{P}$ be the efficiency of A where $P \geq 1$.

Theorem 7. Let $N_P^E = c_{E_P} \cdot r_{E_P}$ denote the dimension of the execution matrix of $A_{k,P}$, it holds that

$$Ef(A_{k,P}) = \frac{\beta^{calc}_{sum,M_{E_{k,1}}}}{N_P^E \cdot R^{calc}(A_{k,P})}. \tag{56}$$

Proof. Since $c_E = P$, it follows that

$$Ef(A_{k,P}) = \frac{Sp(A_{k,P})}{P} = \frac{\beta^{calc}_{sum,M_{E_{k,1}}}}{c_{E_P} \cdot r_{E_P} \cdot R^{calc}(A_{k,P})}. \tag{57}$$

□

Definition 27 (Ideal Efficiency in $\hat{\varrho}(A_{k,P})$)**.** Since $Sp_{(k,P)} \leq P \cdot \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}$, it always is $Ef(A_{k,P}) \leq \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}$. So let

$$Ef_{Ideal}(A_{k,P}) = \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})} \tag{58}$$

be the ideal efficiency of $A_{k,P}$.

Remark 19. It is worth to note the role of parameters $R^{calc}(A_{k,P})$ and $R^{calc}(A_{k,1})$ in (47), (55) and (56). If in $A_{k,P}$ there are few operators which are much more time consuming than the others, and $k >> r_E$ then $\beta^{calc}_{M_{E_{k,P}}} \simeq \beta^{calc}_{sum,M_{E_{k,1}}}$ and $R^{calc}(A_{k,P}) >> R^{calc}(A_{k,1})$. The more the operators are and the greater the difference is in (55), or the lower the ratio is in (47) and (56). Hence, the greater the

overhead is, the lower the speed up and the efficiency are. This is a consequence of a problem decomposition, associated to $A_{k,P}$ not well balanced.

Let us now suppose that the algorithm $A_{k,P}$ is perfectly parallel, that is its execution matrix M_{E_P} has not any empty element. Since $r_{E_P} \cdot c_{E_P} = C(A_{k,P})$ it follows from Corollary 3 that

$$Sp(A_{k,P}) = Sp_{Ideal}(A_{k,P}) = P \cdot \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})},$$

from (53) that

$$Oh(A_{k,P}) = Oh_{Ideal}(A_{k,P}) = \big(C(A_{k,P}) \cdot (R^{calc}(A_{k,P}) - R^{calc}(A_{k,1}))\big) \cdot tcalc,$$

from (58)

$$Ef(A_{k,P}) = Ef_{Ideal}(A_{k,P}) = \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}.$$

Remark 20. If $P = c_D$, $r_E = r_D$ and $c_E = c_D$, if $P = c_D$ then the following results hold on:

1. $Q(A_{k,P}) = c_D \cdot r_D \cdot R^{calc}(A_{k,P}) \cdot tcalc = N_D \cdot R^{calc}(A_{k,P}) \cdot tcalc$;
2. $Sp(A_{k,P}) = \frac{C(A_{k,P})}{r_D} \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}$;
3. $Oh(A_{k,P}) = (c_D \cdot r_D - C(A_{k,P})) \cdot R^{calc}(A_{k,P}) \cdot tcalc$;
4. $Ef(A_{k,P}) = \frac{C(A_{k,P})}{r_D \cdot c_D} \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}$.

Example 2. The well known reduction problem is interesting to expose the nature of Algorithm Overhead Oh and its impact in some classical metrics.

Let $\mathcal{B}_{27}$ denote the computational problem of the sum of 27 real numbers and $D_{13}(\mathcal{B}_{27}) = \{\mathcal{B}_3^i\}_{0 \le i < 13} \in \mathcal{DB}_{27}$ one of its decompositions, where $\mathcal{B}_3^i$ represents the sum of 3 real numbers.

The decomposition matrix is

$$M_D(D_{13}(\mathcal{B}_{27}) = \begin{bmatrix} \mathcal{B}_3^0 & \mathcal{B}_3^1 & \mathcal{B}_3^2 & \mathcal{B}_3^3 & \mathcal{B}_3^4 & \mathcal{B}_3^5 & \mathcal{B}_3^6 & \mathcal{B}_3^7 & \mathcal{B}_3^8 \\ \mathcal{B}_3^9 & \mathcal{B}_3^{10} & \mathcal{B}_3^{11} & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \mathcal{B}_3^{12} & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \end{bmatrix}. \tag{59}$$

Therefore, the concurrency degree is $c_{D_{13}} = 9$, the dependency degree is $r_{D_{13}} = 3$, and the problem is not perfectly decomposed. Let us suppose $\mathcal{B}_{27}$ is solvable on $\mathcal{M}_P$ with $P = 3$. Let $Cop_{\mathcal{M}_3} = \{++, \ldots\}$, be the computing operators of $\mathcal{M}_3$, and let $A_{D_{13}(\mathcal{B}_{27}),\mathcal{M}_3} = \{++_0, \ldots, ++_{12}\}$ be the algorithm that we choose to solve $\mathcal{B}_{27}$, given

$D_{13}(\mathcal{B}_{27})$. Then the execution matrix is

$$M_{E_{13,3}} = \begin{bmatrix} ++_0 & ++_1 & ++_2 \\ ++_3 & ++_4 & ++_5 \\ ++_6 & ++_7 & ++_8 \\ ++_9 & ++_{10} & ++_{11} \\ ++_{12} & \varnothing & \varnothing \end{bmatrix} \tag{60}$$

and the corresponding algorithm is simply parallel. Moreover, $T(A_{13,3}) = 5 \cdot tcalc$, where $tcalc$ is the execution time for the sum $++$ of three real numbers.

If we take another $\mathcal{M}_P$ with $P = 4$ where $\mathcal{B}_{27}$ is solvable, let us suppose that $Cop_{\mathcal{M}_4} = Cop_{\mathcal{M}_3}$ and $A_{D_{13}(\mathcal{B}_{27}),\mathcal{M}_4} = A_{D_{13}(\mathcal{B}_{27}),\mathcal{M}_3}$. Then, the execution matrix can be written as

$$M_{E_{13,4}} = \begin{bmatrix} ++_0 & ++_1 & ++_2 & ++_3 \\ ++_4 & ++_5 & ++_6 & ++_7 \\ ++_8 & \varnothing & \varnothing & \varnothing \\ ++_9 & ++_{10} & ++_{11} & \varnothing \\ ++_{12} & \varnothing & \varnothing & \varnothing \end{bmatrix}. \tag{61}$$

The corresponding algorithm is still simply parallel, and $T(A_{13,4}) = 5 \cdot tcalc$.

Considering the classical metrics of speed-up and efficiency, evaluated together with cost and algorithm overhead, we have:

- for $A_{D_{13}(\mathcal{B}_{27}),\mathcal{M}_3}$

$$\begin{aligned} Sp(A_{13,3}) &= \frac{r_{E_{A_{13,1}}}}{r_{E_{A_{13,3}}}} = 2.6, \\ Q(A_{13,3}) &= c_{E_{A_{13,3}}} \cdot r_{E_{A_{13,3}}} \cdot tcalc = 15 \cdot tcalc, \\ Oh(A_{13,3}) &= Q(A_{13,3}) - Q(A_{13,1}) = 2 \cdot tcalc, \\ Ef(A_{13,3}) &= \frac{r_{E_{A_{13,1}}}}{r_{E_{A_{13,3}}} \cdot c_{E_{A_{13,3}}}} = 0.87, \end{aligned} \tag{62}$$

- for $A_{D_{13}(\mathcal{B}_{27}),\mathcal{M}_4}$

$$\begin{aligned} Sp(A_{13,4}) &= \frac{r_{E_{A_{13,1}}}}{r_{E_{A_{13,4}}}} = 2.6, \\ Q(A_{13,4}) &= c_{E_{A_{13,4}}} \cdot r_{E_{A_{13,4}}} \cdot tcalc = 20 \cdot tcalc, \\ Oh(A_{13,4}) &= Q(A_{13,3}) - Q(A_{13,1}) = 7 \cdot tcalc, \\ Ef(A_{13,4}) &= \frac{r_{E_{A_{13,1}}}}{r_{E_{A_{13,4}}} \cdot c_{E_{A_{13,4}}}} = 0.65. \end{aligned} \tag{63}$$

Observe that while the speed up is the same, the overhead reveals that the mapping on M_4 is not the optimal one, showing the performance bottleneck of the algorithm.

6 ALGORITHMS WHOSE OPERATORS HAVE THE SAME EXECUTION TIME

We assume that all the operators of the algorithm have the same execution time. For example they are the elementary floating point operations. The execution time is $\beta^{calc} \cdot tcalc$, and without loss of generality we assume that $\beta^{calc} = 1$.

Hence, it follows that, $\forall P, \quad \beta^{calc}_{r,M_{E_{k,P}}} = 1, \beta^{calc}_{M_{E_{k,P}}} = r_E, \beta^{calc}_{sum,M_{E_{k,P}}} = k$. Finally, from (25) it follows that $\forall P, R^{calc}(A_{k,P}) = 1$. Hence, we get

- $Sp(A_{k,P}, A_{k',1}) := \frac{k'}{k} \cdot \frac{k}{r_E}$,
- if $Q = 1$, then $Sp(A_{k,P}) := \frac{k}{r_E}$,
- $Sp_{Ideal}(A_{k,P}, A_{k',1}) = Sc_{up}(A_{k,P}, A_{k',1}) \cdot P = \frac{k'}{k} \cdot P$,
- $Sp_{Ideal}(A_{k,P}) = c_{E_P} = P$.

Finally, if $\mathcal{B}_{N_r}$ is perfectly decomposed, then

$$T(A_{k,P}) = \frac{T(A_{k,1})}{P}, \tag{64}$$

i.e., $A_{k,P}$ has the ideal speed up in the classical definition.

Let us now consider matrices $M_{E_{k,P}}$ associated with algorithms in $\hat{\varrho}(A_{k,P})$, varying P. The following results hold: $Q(A_{k,P}) = c_E \cdot r_E \cdot tcalc$ and, if $c_E = 1$, then $Q(A_{k,1}) = k \cdot tcalc$; $Oh_{Ideal}(A_{k,P}) = 0$; $Ef_{Ideal}(A_{k,P}) = 1$.

Theorem 8. Let us suppose that

$$\forall I^{i_j} \in Cop_{\mathcal{M}_P}, \quad t_{i_j} = \beta^{calc}_{i_j,M_{E_{k,1}}} \cdot tcalc = tcalc, \quad \forall i, j. \tag{65}$$

Given $A_{k,P}$, $P > 1$, $M_{E_{k,P}}$ of order $N_P^E = r_E \cdot P$, let V_r be the number of empty elements of the row r of $M_{E_{k,P}}$; it is

$$Oh(A_{k,P}) = \sum_{r=0}^{r_E-1} V_r \cdot tcalc. \tag{66}$$

Proof. It holds that $c_E \cdot r_E = card(A_{k,P}) + \sum_{r=0}^{r_E-1} V_r = C(A_{k,P}) + \sum_{r=0}^{r_E-1} V_r = k + \sum_{r=0}^{r_E-1} V_r$ then from (52)

$$Oh(A_{k,P}) = \left(k + \sum_{r=0}^{r_E-1} V_r - k \right) \cdot tcalc = \sum_{r=0}^{r_E-1} V_r \cdot tcalc. \tag{67}$$

□

Remark 21. Note that $\sum_{r=0}^{r_E-1} V_r$ is the sparsity degree of the execution matrix.

Among the decomposition approaches, recursive decomposition very often is the most suitable approach for employing a performance analysis, especially in the presence of complex algorithms solving real-world applications/simulations. In this case, as described in the toy example below, the problem is solved by firstly decomposing it into a set of independent sub-problems. Furthermore, each one of these subproblems is solved by applying a similar decomposition into smaller subproblems followed by a combination of their results, and so on. In this way we get a decomposition matrix whose elements are problems which could be subsequently decomposed and analyzed until the desired level of detail is reached.

Example 3. Let $\mathcal{B}_{16}$ denote the computational problem of the sum of 16 real numbers and $D_3(\mathcal{B}_{16}) = \{B_8, B_8, B_2\} \in \mathcal{DB}_{16}$. The decomposition matrix is

$$M_{D_3}(\mathcal{B}_{16}) = \begin{bmatrix} \mathcal{B}_8 & \mathcal{B}_8 \\ \mathcal{B}_2 & \varnothing \end{bmatrix}. \tag{68}$$

If $\mathcal{B}_8$ can be decomposed as $D_3^1(\mathcal{B}_8) = \{\mathcal{B}_4, \mathcal{B}_4, \mathcal{B}_2\} \in \mathcal{DB}_8$ then

$$M_{D_3^1}(\mathcal{B}_8) = \begin{bmatrix} \mathcal{B}_4 & \mathcal{B}_4 \\ \mathcal{B}_2 & \varnothing \end{bmatrix}. \tag{69}$$

In the same way, if $\mathcal{B}_4$ can be decomposed as $D_3^2(\mathcal{B}_4) = \{\mathcal{B}_2, \mathcal{B}_2, \mathcal{B}_2\} \in \mathcal{DB}_8$ and

$$M_{D_3^2}(\mathcal{B}_4) = \begin{bmatrix} \mathcal{B}_2 & \mathcal{B}_2 \\ \mathcal{B}_2 & \varnothing \end{bmatrix}. \tag{70}$$

We have three decompositions for $\mathcal{B}_{16}$:

$$\begin{aligned}
D_3 \in \mathcal{D}(\mathcal{B}_{16}) &= \{\mathcal{B}_8, \mathcal{B}_8, \mathcal{B}_2\}, \\
D_7 \in \mathcal{D}(\mathcal{B}_{16}) &\equiv D_3^1(\mathcal{B}_8) \cup D_3^1(\mathcal{B}_8) \cup \{\mathcal{B}_2\} \\
&\equiv \{\mathcal{B}_4, \mathcal{B}_4, \mathcal{B}_2\} \cup \{\mathcal{B}_4, \mathcal{B}_4, \mathcal{B}_2\} \cup \{\mathcal{B}_2\}, \\
D_{15} \in \mathcal{D}(\mathcal{B}_{16}) &\equiv D_3^2(\mathcal{B}_4) \cup D_3^2(\mathcal{B}_4) \cup \{\mathcal{B}_2\} \cup D_3^2(\mathcal{B}_4) \cup D_3^2(\mathcal{B}_4) \cup \{\mathcal{B}_2\} \cup \{\mathcal{B}_2\} \\
&\equiv \{\mathcal{B}_2, \mathcal{B}_2, \mathcal{B}_2\} \cup \{\mathcal{B}_2, \mathcal{B}_2, \mathcal{B}_2\} \cup \{\mathcal{B}_2\} \cup \{\mathcal{B}_2, \mathcal{B}_2, \mathcal{B}_2\} \\
&\quad \cup \{\mathcal{B}_2, \mathcal{B}_2, \mathcal{B}_2\} \cup \{\mathcal{B}_2\} \cup \{\mathcal{B}_2\} \\
&\equiv \{\mathcal{B}_2^i\}_{0 \le i < 15} \in \mathcal{DB}_{16} \qquad (71)
\end{aligned}$$

with the following characteristics, according to the corresponding decomposition matrices:

- D_3: cardinality 3, concurrency degree 2 and dependence degree 2,
- D_7: cardinality 7, concurrency degree 4 and dependence degree 3,
- D_{15}: cardinality 15, concurrency degree 8 and dependence degree 4,

meaning that the intrinsic concurrency of a problem heavily depends on the decomposition chosen for that problem. Each decomposition has a level of detail depending on the type of subproblems that are considered.

7 SOFTWARE

From now on, we consider memory accesses performed by an algorithm and we assume, for simplicity, that to each access corresponds one read/write of a single data. Moreover, we assume that computations and memory accesses are not performed simultaneously, instead they depend on each other.

Definition 28. Given the set of elementary operators of $\mathcal{M}_P$, we introduce memory access operators corresponding to the memory access (read/write) of processing elements of $\mathcal{M}_P$. The set $OA_{\mathcal{M}_P} = \{r(\cdot), w(\cdot)\}$ where $r(a)$, which reads a, and $w(a)$, which writes a, contains memory access operators of $\mathcal{M}_P$.

Note that $\mathcal{M}_P$ is now the union of the set of elementary operators and the set of memory accesses operators, $\mathcal{M}_P = Op_{\mathcal{M}_P} \bigcup OA_{\mathcal{M}_P}$.

Definition 29. We introduce the ordered set (whose elements should not be different) of accesses operators of $\mathcal{M}_P$ $AC = \{oa_0(\cdot), oa_1(\cdot), \ldots, oa_k(\cdot)\}$ where $oa_i(\cdot) \in OA_{\mathcal{M}_P}$. Moreover, we consider the surjective correspondence

$$\bar{\gamma} : oa_i(\cdot) \in OA_{\mathcal{M}_P} \longleftrightarrow op_i \in A_{k,P}. \qquad (72)$$

Note that $card(AC) \geq card(A_{k,P})$.

The set $AC_{\mathcal{M}_{P,L,nd_L}}(l) := \{oa^l_{i_0}(\cdot), oa^l_{i_1}(\cdot), \ldots, oa^l_{i_k}(\cdot)\} \subset Xop_{\mathcal{M}_P}$ denotes an ordered set of accesses operators of $\mathcal{M}_{P,L,nd_L}$ at level l. Let $AC_{\mathcal{M}_{P,L,nd_L}} := \bigcup_{l=1}^{L} AC_{\mathcal{M}_{P,L,nd_L}}(l)$ denote the set of the memory accesses of $\mathcal{M}_{P,L,nd_L}$. For simplicity of notations, if there is no ambiguity, the set $AC_{\mathcal{M}_{P,L,nd_L}}$ of memory accesses of $\mathcal{M}_{P,L,nd_L}$ is briefly denoted as $AC(L)$.

Definition 30 (Software)**.** The set $SW(A_{k,P}) := A_{k,P} \cup AC(L)$ where the order relation on $AC(L)$ is induced by the ordering on $A_{k,P}$ is said the Software corresponding to algorithm $A_{k,P}$. More simply, in the sequel we denote the Software as $SW(A_{k,P})$.

Definition 31. Given M_E and AC, matrix $AM_{A_{D_k},\mathcal{M}_{P,L,nd_L}}(l)$, defined in AC of order $r_{AM_l} \times c_{AM_l}$, with $c_{AM} = nd_l$[11] is said the l^{th} access matrix of SW.

Let $r_{AM} := \sum_{l=1}^{L} r_{AM_l}$ and let $r_{COM} := r_{AM_L}$ $(r_{COM} \leq r_{AM})$ be the parameter counting the rows of the L^{th} matrix $AM(L)$ related to L^{th} level. If $P = 1$ then $r_{COM} = 0$.

Definition 32 (Memory Access Time)**.** The quantity

$$T_M(SW(A_{k,P})) := \sum_{l=1}^{L-1} (r_{AM_l} \cdot tmem_l) \tag{73}$$

is said memory access time of $SW(A)$.

Computational intensity is defined as the number of operations per memory accesses [23]. More precisely, it measures how intensely A computes with data, once it has been received.

Definition 33 (Computational Intensity)**.** The quantity

$$C_I(SW(A_{k,P})) := \frac{r_E}{r_{AM}} \in [1, \infty[\tag{74}$$

is said software computational intensity.

Remark 22. If instead of r_{AM} we only consider the number of rows of the L^{th} memory access matrix, which is related to the software communications, i.e., we only consider r_{COM} and take the reciprocal of $C_I(SW(A_{k,P}))$, we get the so called software communication intensity. It measures how much communications dominate with respect to the operations. This quantity is usually called surface-to-volume ratio [14].

[11] In general $c_{AM_l} \leq nd_l$, but with no loss of generality we assume that $c_{AM_l} = nd_l$.

Definition 34 (Communication Intensity)**.** The quantity

$$Com_I(SW(A_{k,P})) := \frac{r_{COM}}{r_E} \tag{75}$$

is said software communication intensity.

Definition 35 (Communication Time)**.** The quantity

$$T_{COM}(SW(A_{k,P})) := r_{COM} \cdot tcom \tag{76}$$

is said the software communication time.

We now assume that $\mathcal{M}_{P,nd_L}$ is such that $P \geq 1$ and $L \geq 3$, that is, it includes the level L of the communications among processing elements. Moreover, since overlapping communication with computation comes at the expense of increased memory requirements, we assume that memory accesses (including communications) and computations cannot be executed simultaneously, but they are dependent on each other.

Definition 36 (Execution Time)**.** The quantity

$$T(SW(A_{k,P})) := T(A_{k,P}) + T_M(SW(A_{k,P})) + T_{COM}(SW(A_{k,P})) \tag{77}$$

is said software execution time of $SW(A_{k,P})$.

Definition 37 (Machine Communication Overhead)**.** The ratio

$$UCom_{oh}(\mathcal{M}_P) := \frac{tcom}{tcalc} \tag{78}$$

is said unitary machine communication overhead.

Observe that at present time it is $UCom_{oh}(\mathcal{M}_{P,L,nd_L}) \gg 1$. Machine communication overhead, also known as *machine balance*, is one of the parameters depending on the machine [17].

Definition 38 (Software Communication Overhead)**.** The quantity

$$Com_{oh}(SW(A_{k,P})) := \frac{T_{COM}(SW(A_{k,P}))}{T(A_{k,P})}, \tag{79}$$

which is the software communication overhead, is expressed as follows

$$\begin{aligned} Com_{oh}(SW(A_{k,P})) &:= \frac{r_{COM}}{r_E} \cdot \frac{tcom}{tcalc} \\ &\equiv Com_I(SW(A_{k,P})) \cdot UCom_{oh}(\mathcal{M}_P). \end{aligned} \tag{80}$$

Definition 39 (Memory Traffic)**.** The quantity

$$M_T(A_{k,P}) := \frac{T_M(SW(A_{k,P}))}{T(A_{k,P})}, \tag{81}$$

which is the memory traffic, is expressed as follows

$$M_T(A_{k,P}) := \frac{\sum_{l=1}^{L-1}(r_{AM_l} \cdot tmem_l)}{\sum_{r=0}^{r_E-1} T_r(A_{k,P})}. \tag{82}$$

Remark 23. If memory traffic grows, then the computational intensity $C_I(SW(A_{k,P}))$ decreases (see Definition 33).

Definition 40 (l^{th} Software Memory Traffic)**.** The ratio

$$M_T(A_{k,P})_l := \frac{r_{AM_l} \cdot tmem_l}{\sum_{r=0}^{r_E-1} T_r(A_{k,P})} \tag{83}$$

is said level the l^{th} memory traffic of $SW(A)$.

Definition 41 (Software Speed Up)**.** Given $SW(A_{k,P_1})$ and $SW(A_{k,P_2})$, where $P_2 > P_1$, the ratio

$$Sp(SW(A_{k,P_2})) := \frac{T(SW(A_{k,P_1}))}{T(SW(A_{k,P_2}))} \tag{84}$$

is said software speed up of $SW(A_{k,P_2})$.

Proposition 2.

$$\begin{aligned} Sp(SW_{A_{k,P},\mathcal{M}_{P,nd_{L1}}}) &= \frac{T(SW_{A_{k,1},\mathcal{M}_{1,nd_{L^1}}})}{T(SW_{A_{k,P},\mathcal{M}_{P,nd_{L^P}}})} \\ &= \frac{r_{E^1} \cdot tcalc + (r_{AM^1} - r_{COM^1}) \cdot tmem + r_{COM^1} \cdot tcom}{r_{E^P} \cdot tcalc + (r_{AM^P} - r_{COM^P}) \cdot tmem + r_{COM^P} \cdot tcom}. \end{aligned}$$

In the same way as we have previously done for algorithm A, the Software efficiency $Ep(SW)$ and all the other performance metrics can be defined in terms of the Software execution time $T(SW)$.

8 CONCLUSIONS

This paper targets an important topic of current importance in High Performance Computing community, which is the performance analysis of parallel algorithms; it should be re-evaluated to find out the best-practice algorithm on novel architectures [4, 15, 16, 18, 20, 21, 26, 34]. In this paper we presented a mathematical framework which can be used to get a multilevel description of a parallel algorithm,

and we proved that it can be suitable for analysing the mapping of an algorithm on a given machine. The model is multilevel, in the sense that it allows the choice of a level of abstraction of both the problem decomposition and of the operations in the algorithm, which determines the level of granularity of the performance analysis. This feature can be very useful in practice to analyze performance of complex algorithms solving real problems and to indicate performance bottlenecks within the algorithm. Furthermore, the model allows to take into account the initial decomposition of the problem into subproblems, and so their mutual dependencies. In order to show how to use the performance model, we validated this approach in practice using real problems on real architectures. In [27], a preliminary performance analysis, carried out considering the speed up of the algorithm before its mapping on the computing architectures, provided the best number of processing elements and grid levels to address the scaling of a multigrid in time algorithm. According to the time-stepping procedure, the performance analysis was carried out choosing matrix-vector products or linear system solutions as the elements of the dependency matrix, and therefore as computing operators of the algorithm. In [3, 14], the performance analysis of the algorithm, carried out in terms of scale up, suggested the introduction of a highly scalable decomposition of a variational data assimilation problem. This approach completely redesigned the mapping of the numerical algorithm on high performance computing architectures.

We remark that energy consumption on computer systems has emerged as an important concern, and the energy consumed in executing an algorithm cannot be inferred from its performance alone. We will employ the proposed framework to also model energy consumption of parallel algorithms. As example, according to [13], we performed the energy analysis of a variational data assimilation algorithm running on an ARM-based HPC systems assuming that total energy depends on the energy consumed in all steps executed by the parallel algorithm; the energy consumed at each step is measured in terms of parameters depending on both the algorithm and the computing architecture [2]. This approach can be used to extend the performance metrics to address the analysis of software energy consumption.

We conclude that we have assumed abstract models for both algorithms and architectures, and we have made numerous simplifying assumptions. Indeed, we believe that a simplified parameterized model gives a useful generalization for a better understanding of algorithms that can run really fast, no matter how complicated the underlying computer architecture [17].

REFERENCES

[1] Amdahl, G. M.: Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. AFIPS Conference Proceedings (AFIPS '67), Vol. 30, 1967, pp. 483–485, doi: 10.1145/1465482.1465560.

[2] Arcucci, R.—Basciano, D.—Cilardo, A.—D'Amore, L.—Mantovani, F.: Energy Analysis of a 4D Variational Data Assimilation Algorithm and Evaluation

on ARM-Based HPC Systems. In: Wyrzykowski, R., Dongarra, J., Deelman, E., Karczewski, K. (Eds.): Parallel Processing and Applied Mathematics (PPAM 2017). Springer, Cham, Lecture Notes in Computer Science, Vol. 10778, 2018, pp. 37–47, doi: 10.1007/978-3-319-78054-2_4.

[3] Arcucci, R.—D'Amore, L.—Celestino, S.—Laccetti, G.—Murli, A.: A Scalable Numerical Algorithm for Solving Tikhonov Regularization Problems. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Kitowski, J., Wiatr, K. (Eds.): Parallel Processing and Applied Mathematics. Springer, Cham, Lecture Notes in Computer Science, Vol. 9574, 2016, pp. 45–54, doi: 10.1007/978-3-319-32152-3_5.

[4] Ballard, G.—Demmel, J.—Holtz, O.—Schwartz, O.: Minimizing Communication in Numerical Linear Algebra. SIAM Journal on Matrix Analysis and Applications, Vol. 32, 2011, No. 3, pp. 866–901, doi: 10.1137/090769156.

[5] Berman, F.—Snyder, L.: Mapping Parallel Algorithms into Parallel Architectures. Journal of Parallel and Distributed Computing, Vol. 4, 1987, No. 5, pp. 439–458, doi: 10.1016/0743-7315(87)90018-9.

[6] Berman, F.: The Mapping Problem in Parallel Computation. In: Rice, J. R. (Ed.): Mathematical Aspects of Scientific Software. Springer, New York, NY, The IMA Volumes in Mathematics and Its Applications, Vol. 14, 1988, pp. 41–57, doi: 10.1007/978-1-4684-7074-1_2.

[7] Bernstein, A. J.: Analysis of Programs for Parallel Processing. IEEE Transactions on Electronic Computers, Vol. EC-15, 1966, No. 5, pp. 757–763, doi: 10.1109/pgec.1966.264565.

[8] Bokhari, S. H.: On the Mapping Problem. IEEE Transactions on Computers, Vol. C-30, 1981, No. 3, pp. 207–214, doi: 10.1109/tc.1981.1675756.

[9] Browne, S.—Dongarra, J.—Garner, N.—Ho, G.—Mucci, P.: A Portable Programming Interface for Performance Evaluation on Modern Processors. The International Journal of High Performance Computing Applications, Vol. 14, 2000, No. 3, pp. 189–204, doi: 10.1177/109434200001400303.

[10] Bosilca, G.—Bouteiller, A.—Danalis, A.—Herault, T.—Lemarinier, P.—Dongarra, J.: DAGuE: A Generic Distributed DAG Engine for High Performance Computing. Parallel Computing, Vol. 38, 2012, No. 1–2, pp. 37–51, doi: 10.1016/j.parco.2011.10.003.

[11] Bosilca, G.—Bouteiller, A.—Danalis, A.—Faverge, M.—Herault, T.—Dongarra, J.: PaRSEC: Exploiting Heterogeneity to Enhance Scalability. Computing in Science and Engineering, Vol. 15, 2013, No. 6, pp. 36–45, doi: 10.1109/mcse.2013.98.

[12] Coffman, E. G. Jr.—Denning, P. J.: Operating Systems Theory. Prentice Hall, 1973.

[13] Korthikanti, V. A.—Agha, G.: Energy-Performance Trade-Off Analysis of Parallel Algorithms for Shared Memory Architectures. Sustainable Computing: Informatics and Systems, Vol. 1, 2011, No. 3, pp. 167–176, doi: 10.1016/j.suscom.2011.05.004.

[14] D'Amore, L.—Arcucci, R.—Carracciuolo, L.—Murli, A.: A Scalable Approach for Variational Data Assimilation. Journal of Scientific Computing, Vol. 61, 2014, No. 2, pp. 239–257, doi: 10.1007/s10915-014-9824-2.

[15] D'Amore, L.—Laccetti, G.—Romano, D.—Scotti, G.—Murli, A.: Towards a Parallel Component in a GPU-CUDA Environment: A Case Study with the L-BFGS Harwell Routine. International Journal of Computer Mathematics, Vol. 92, 2014, No. 1, pp. 59–76, doi: 10.1080/00207160.2014.899589.

[16] D'Amore, L.—Mele, V.—Laccetti, G.—Murli, A.: Mathematical Approach to the Performance Evaluation of Matrix Multiply Algorithm. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Kitowski, J., Wiatr, K. (Eds.): Parallel Processing and Applied Mathematics. Springer, Cham, Lecture Notes in Computer Science, Vol. 9574, 2016, pp. 25–34, doi: 10.1007/978-3-319-32152-3_3.

[17] Demmel, J.: Applications of Parallel Computers. U.C. Berkeley CS267, `http://www.cs.berkeley.edu/~demmel/cs267_Spr12`, 2012.

[18] Demmel, J.—Eliahu, D.—Fox, A.—Kamil, S.—Lipshitz, B.—Schwartz, O.—Spillinger, O.: Communication-Optimal Parallel Recursive Rectangular Matrix Multiplication. Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing (IPDPS '13), 2013, pp. 261–272, doi: 10.1109/ipdps.2013.80.

[19] Flatt, H. P.—Kennedy, K.: Performance of Parallel Processors. Parallel Computing, Vol. 12, 1989, No. 1, pp. 1–20, doi: 10.1016/0167-8191(89)90003-3.

[20] Gunnels, J. A.—Henry, G. M.—van de Geijn, R. A.: A Family of High-Performance Matrix Multiplication Algorithms. In: Alexandrov, V. N., Dongarra, J. J., Juliano, B. A., Renner, R. S., Tan, C. J. K. (Eds.): Computational Science – ICCS 2001, Part I. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2073, 2001, pp. 51–60, doi: 10.1007/3-540-45545-0_15.

[21] Gunnels, J. A.—Gustavson, F. G.—Henry, G. M.—van de Geijn, R. A.: FLAME: Formal Linear Algebra Methods Environment. ACM Transactions on Mathematical Software, Vol. 27, 2001, No. 4, pp. 422–455, doi: 10.1145/504210.504213.

[22] Gupta, A.—Kumar, V.: Performance Properties of Large Scale Parallel Systems. Journal of Parallel and Distributed Computing, Vol. 19, 1993, No. 3, pp. 234–244, doi: 10.1006/jpdc.1993.1107.

[23] Hockney, R. W.: The Science of Computer Benchmarking. SIAM, Software, Environments and Tools Series, 1996, doi: 10.1137/1.9780898719666.

[24] Kronsjö, L. I.: Algorithms, Their Complexity and Efficiency. John Wiley and Sons, New York, NY, USA, 1979.

[25] Kuck, D. J.—Kuhn, R. H.—Padua, D. A.—Leasure, B.—Wolfe, M.: Dependence Graphs and Compiler Optimization. Proceeding of the 8th ACM SIGPLAN – SIGACT Symposium on Principles on Programming Languages (POPL '81), 1981, pp. 207–218, doi: 10.1145/567532.567555.

[26] Miller, B.—Vahid, F.—Givargis, T.—Brisk, P.: Graph-Based Approaches to Placement of Processing Element Networks on FPGAs for Physical Model Simulation. ACM Transaction on Reconfigurable Technology and Systems, Vol. 7, 2015, No. 4, Art. No. 37, doi: 10.1145/2629521.

[27] Mele, V.—Constantinescu, E. M.—Carracciuolo, L.—D'Amore, L.: A PETSc Parallel-in-Time Solver Based on MGRIT Algorithm. Concur-

rency and Computation: Practice and Experience, Vol. 30, 2018, No. 24, doi: 10.1002/cpe.4928.

[28] Moldovan, D. I.: On the Analysis and Synthesis of VLSI Algorithms. IEEE Transactions on Computers, Vol. C-31, 1982, No. 11, pp. 1121–1126, doi: 10.1109/tc.1982.1675929.

[29] Rico-Gallego, J. A.—Díaz-Martín, J. C.—Manumachu, R. R.—Lastovetsky, A. L.: A Survey of Communications Performance Models for High-Performance Computing. ACM Computing Surveys, Vol. 51, 2019, No. 6, Art. No. 126, doi: 10.1145/3284358.

[30] Sharp, J. A. (Ed.): Data Flow Computing: Theory and Practice. Ablex Publishing Corporation, Norwood, New Jersey, 1992.

[31] Lee, S.-Y.—Aggarwal, J. K.: A Mapping Strategy for Parallel Processing. IEEE Transactions on Computers, Vol. C-36, 1987, No. 4, pp. 433–442, doi: 10.1109/tc.1987.1676925.

[32] Tekinerdogan, B.—Arkin, E.: Architectural Framework for Mapping Parallel Algorithms to Parallel Computing Platforms. 2nd International Workshop on Model Driven Engineering for High Performance and Cloud Computing (MDHPCL 2013), Miami, Florida, CEUR Workshop Proceedings, Vol. 1118, 2013, pp. 53–62.

[33] Tjaden, G. S.—Flynn, M. J.: Detection and Parallel Execution of Independent Instructions. IEEE Transactions on Computers, Vol. C-19, 1970, No. 10, pp. 889–895, doi: 10.1109/t-c.1970.222795.

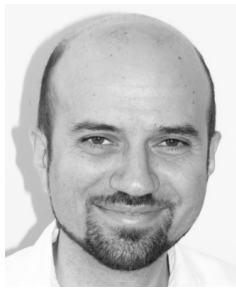
[34] Voevodin, V. V.—Antonov, A. S.—Dongarra, J. J.: AlgoWiki: An Open Encyclopedia of Parallel Algorithmic Features. Supercomputing Frontiers and Innovations, Vol. 2, 2015, No. 1, pp. 4–18, doi: 10.14529/jsfi150101.

[35] Zhong, Z.—Rychkov, V.—Lastovetsky, A.: Data Partitioning on Multicore and Multi-GPU Platforms Using Functional Performance Models. IEEE Transactions on Computers, Vol. 64, 2015, No. 9, pp. 2506–2518, doi: 10.1109/tc.2014.2375202.

Luisa D'Amore received her degree in mathematics in 1988 and her Ph.D. in applied mathematics and computer science in 1995. Since 1996 she works as researcher in numerical analysis and since 2001 she is Associate Professor of numerical analysis. Since 2011 she has been working as an associate staff researcher of the ASC (Advanced Scientific Computing) Division of CMCC (Euro Mediterranean Center on Climate Changes). She is a member of the Academic Board for the Ph.D. in mathematics and informatics at the University of Naples Federico II and a teacher of courses in numerical analysis, scientific computing and parallel computing. Her research activity focuses on scientific computing and it is addressed to the numerical solution of ill-posed inverse problems with applications in image analysis, medical imaging, astronomy, digital restoration of films and data assimilation. The need of computing the numerical solution in a suitable time, induced by the applications, often requires the use of advanced computing architectures.

This involves designing and development of algorithms and software capable of exploiting the high performance of emergingcomputing infrastructures. She is (co)author of about 100 publications in refereed journals and conference proceedings.

Valeria Mele is a Researcher at University of Naples Federico II (Naples, Italy). She obtained a degree in informatics and a Ph.D. in computational science, and for many years has been working as teaching assistant in Parallel and Distributed Computing classes at University of Naples Parthenope and University of Naples Federico II. Meanwhile, her research activity has been mainly focused on the development and the performance evaluation of parallel algorithms and software for heterogeneous, hybrid and multilevel parallel architectures, from multicore to GPU-enhanced machines and modern clusters and supercomputers. After attending the Argonne Training Program on Extreme-Scale Computing (ATPESC) and visiting the Argonne National Laboratory (ANL, Chicago, Illinois, USA), she is now mainly working on the designing, the implementation and the performance prediction/evaluation of software with/for the PETSc (Portable, Extensible Toolkit for Scientific Computation) library, collaborating with applied mathematicians and computer scientists at the Mathematics and Computer Science Division of the ANL.



Diego Romano obtained the Laurea degree (Italian M.Sc. equivalent) in mathematics in 2000, and his Ph.D. degree in computational and computer sciences from the University of Naples Federico II, Italy, in 2012. He obtained a permanent position as researcher at the Italian National Research Council (CNR) in 2008, where he is currently employed at the Institute for High Performance Computing and Networking (ICAR). His research interests include performance and design of GPU computing algorithms. Within this field, he works, for instance, on the global illumination problem in computer graphics, and on a mathematical model for the performance analysis.

Giuliano Laccetti is Professor of computer science at the University of Naples Federico II, Italy. He obtained his Laurea degree (cum laude) in physics from the University of Naples. His main research interests are mathematical software, high performance architecture for scientific computing, distributed computing, grid and cloud computing, algorithms on emerging hybrid architectures (CPU+GPU, ...), Internet of Things. He has been organizer and chair of several workshops joint to larger international conferences. He is author (or co-author) of about 100 papers published in international refereed journals, books, and conference proceedings.