Original paper

An Innovative Tool for Effectively Applying Highly Parallelized Hardware To Problems of Elasticity

Ismael Herrera* and Iván Contreras

Received: January 17, 2015; accepted: June 06, 2015; published on line: December 31, 205

Resumen

En la actualidad los métodos de descomposición de dominio (DDM, por sus siglas en inglés) más eficientes como instrumento de paralelización son los métodos sin traslape (non-overlapping). Su alta eficiencia es debida a la independencia muy significativa que logran los problemas locales planteados en subdominios que no se traslapan. Sin embargo, los métodos de discretización estándar que habían usado hasta ahora los DDM, aún los sin traslape, utilizan sistemas de nodos en que algunos de ellos son compartidos por varios subdominios de la descomposición. Ésta es una característica limitativa del estado del arte actual de este tipo de procedimientos y, muy probablemente, mayores niveles de independencia de los problemas locales podrían lograrse si se le eliminara. I. Herrera y sus colaboradores han atacado este problema, para lo cual han introducido una nueva manera de formular los DDM que no tiene esta limitación: el método DVS. Un rasgo conspicuo de esta forma de abordar la descomposición de dominio es que se utiliza un método nuevo de discretización de las EDPs, también introducido en la línea de investigación a la que pertenece este artículo, conocido con el nombre de 'discretización sin traslape' (non-overlapping discretization), en el cual cada nodo de la discretización pertenece a uno y solo uno de los subdominios de la descomposición del dominio. Aunque los métodos DVS ya se han desarrollado considerablemente, para que rindan frutos plenamente es indispensable contar con códigos que permitan su implementación eficiente. A eso precisamente está dedicado este artículo: presentar y poner a prueba software de tales características. El software aquí reportado muestra que los algoritmos DVS son los más adecuados para desarrollar software que permita la aplicación efectiva de equipo de cómputo avanzado, altamente en paralelo, a la solución de las ecuaciones diferenciales parciales de los modelos de la ciencia y la ingeniería. Aunque el software que aquí se presenta trata específicamente problemas de elasticidad lineal, los algoritmos DVS son muy eclécticos y pueden ser aplicados a una gran diversidad de ecuaciones diferenciales parciales, después de que las mismas han sido discretizadas. Además, ahora se continúa con trabajo adicional de investigación para desarrollar códigos de propósito general basados en los algoritmos DVS.

Palabras clave: Software en paralelo para EDPs, procesamiento en paralelo de elasticidad, cómputo de alto rendimiento, HPC, elasticidad estática, cómputo en paralelo, métodos de descomposición de dominio.

I. Herrera*
Instituto de Geofísica
Universidad Nacional Autónoma de México
Circuito de la Investigación Científica s/n
Ciudad Universitaria
Delegación Coyoacán, 04510
México D.F., México
**Corresponding autor: iherrerarevilla@gmail.com*

I. Contreras
Posgrado en Ciencia e Ingeniería de la Computación
Universidad Nacional Autónoma de México
Circuito de la Investigación Científica s/n
Ciudad Universitaria
Delegación Coyoacán, 04510
México D.F., México

Abstract

At present, the most efficient domain decomposition methods (DDM) are non-overlapping methods. The improved efficiency of such methods is due to the significant independence achieved by local problems when the subdomains are non-overlapping. However, standard discretizations applied up to now in non-overlapping DDMs use systems of nodes in which some of the nodes are shared by more than one subdomain of the domain decomposition. This is a limiting feature of the present state-of-the-art in these techniques and apparently further increases of the independence of local problems should be expected if this limiting characteristic was eliminated. In previous work, I. Herrera and co-workers have developed a new approach to domain decomposition methods: the 'DVS framework' that addresses this problem introducing a new discretization method, the 'non-overlapping discretization method', in which a non-overlapping system of nodes is used in the discrete formulation of the problem. Although the DVS algorithms have already been developed significantly, to profit from such advances it is essential to have available effective codes that permit their efficient implementation. As a further contribution in this line of research, in this paper we present and test software of such characteristics. The results here reported indicate that the DVS algorithms are very suitable for developing software that permits to apply effectively the most advanced hardware in parallel available at present to the solution of partial differential equations. Although the software here reported specifically treats static elasticity only, the DVS-algorithms are very eclectic and can be applied to a great diversity of problems after they have been discretized. Additional research work is being carried out oriented to develop general purpose codes based on the DVS algorithms.

Key words: Parallel software for PDEs, parallel processing of elasticity, high performance computing, HPC, elastostatics, parallel computing, domain decomposition methods (DDM).

Introduction

Mathematical models occurring in science and engineering, lead to systems of partial differential equations (PDEs) (Herrera and Pinder, 2012), whose solution methods are based on the computational processing of large-scale algebraic systems and the advance of many areas, particularly Earth Sciences, depends on the application of the most powerful computational hardware to them (Presiden'ts Information Technology Advisoty Committee, 2005).

Parallel computing is outstanding among the new computational tools, especially at present when further increases in hardware speed apparently have reached insurmountable barriers.

As it is well known, the main difficulties of parallel computing are associated with the coordination of the many processors that carry out the different tasks and the information-transmission. Ideally, given a task, these difficulties disappear when such *'a task is carried out with the processors working independently of each other'*. We refer to this latter condition as the *'paradigm of parallel-computing software'*.

The emergence of parallel computing prompted on the part of the computational-modeling community a continued and systematic effort with the purpose of harnessing it for the endeavor of solving the mathematical models of scientific and engineering systems. Very early after such an effort began, it was recognized that domain decomposition methods (DDM) were the most effective technique for applying parallel computing to the solution of partial differential equations (DDM Organization 1988-2014), since such an approach drastically simplifies the coordination of the many processors that carry out the different tasks and also reduces very much the requirements of information-transmission between them (Toselli and Widlund, 2005) (Farhat *et al*., 2000).

When a *DDM* is applied, firstly a discretization of the mathematical model is carried out in a *fine-mesh* and, afterwards, a *coarse-mesh* is introduced, which properly constitutes the domain-decomposition. The *'DDM-paradigm'*, a paradigm for domain decomposition methods concomitant with the *paradigm of parallel-computing software* (Herrera *et al*., 2014), consists in *'obtaining the <u>global</u> solution by solving <u>local</u> problems exclusively'* (a <u>local</u> problem is one defined separately in a subdomain of the *coarse-mesh*). Stated in a simplistic manner, the basic idea is that, when the *DDM-paradigm* is satisfied, full parallelization can be achieved by assigning each subdomain to a different processor.

When intensive DDM research began much attention was given to *overlapping DDMs*, but soon after attention shifted to *non-overlapping* DDMs. When the *DDM-paradigm* is taken into account, this evolution seems natural because it is easier to uncouple the local problems when the subdomains do not overlap. However, even in this kind of methods different subdomains are linked by interface nodes that are shared by several subdomains and, therefore, *non-overlapping DDMs* are actually overlapping when seen from the perspective of the nodes used in the discretization. So, a more thorough uncoupling of the local problems and significant computational advantages should be expected if it were possible to carry out the discretization of the differential equations in a *'non-overlapping system of nodes'* (Herrera *et al*., 2014); i.e., a set of nodes with the property that each one of them belongs to one and only one subdomain of the *coarse-mesh* (this is the mesh that constitutes a *domain decomposition*). In (Herrera *et al*., 2014), as in what follows, discretization methods that fulfill these conditions are referred to as *non-overlapping discretizations*.

In a line of research, which this paper belongs to, I. Herrera and co-workers addressed this problem and to cope with it have developed a framework -the *'DVS-framework'*- thoroughly formulated using a *non-overlapping discretization* of the original partial differential equations. Due to the properties of *non-overlapping discretizations* in such algorithms the links between different processors are very much relaxed, and also the required information-transmission between them is reduced. Such properties, as well as preliminary analysis of the algorithms, indicate that they should be extremely adequate to program the treatment of partial differential equations occurring in science and engineering models by the highly parallelized hardware of today. Although the *DVS-algorithms* have already been significantly developed and some examples have been previously treated (Herrera *et al*., 2014 and Carrillo-Ledesma *et al*., 2013), up to now no software that took full advantage of the *DVS-algorithms* had been developed. Clearly, to profit fully from such advances it is essential to develop software, carefully coded, which permit applying effectively the *DVS-algorithms* to problems of interest in science and engineering. As a further contribution to these advances, in this paper, for the first time we present and test software of such characteristics.

Overview of DVS-software

The *derived-vector-space framework (DVS-framework)* deals with the matrix that is obtained after the partial differential equation (PDE), or system of such equations, has been discretized by means of a standard discretization procedure (i.e., an *overlapping discretization*). The resulting discrete system of equations is referred to as the *original-system*.

The DVS-procedures follow the next steps:

1. The partial differential equation, or system of such equations, is discretized by any standard method that satisfies the axioms of the theory (here stated in section how to build non-overlapping discretizations) in a mesh -called the *fine-mesh*- to obtain a discrete problem that is written as

$$\underline{\underline{M}}\,\underline{U} = \underline{F} \tag{2.1}$$

This is called the *original-problem*, while the nodes of the *fine-mesh* are called *original-nodes*. The notation $\widehat{X}$ will be used for the whole set of *original-nodes*; any function defined on the set $\widehat{X}$ by definition is an *original-vector*. Finally, the notation $\widehat{W}$ will be used for the linear space spanned by the *original-vectors*, which in turn is called *original-vector space*;

2. A *coarse-mesh* is introduced, which constitutes a non-overlapping decomposition of the problem-domain. The system of *original-nodes* turns out to be *overlapping* with respect to the *coarse-mesh*;

3. A system of *non-overlapping nodes* (the *derived-nodes*), denoted by X, is constructed applying the procedure explained in previous articles (see also preliminary notions and notations). The functions defined in the whole set X are by definition the *derived-vectors* and the notation W is used for the whole linear space of *derived-vectors*, which constitutes the *derived-vector space*;

4. The theory of the *DVS-framework* supplies a formula that permits transforming the *original-discretization* into a *non-overlapping discretization*. Applying this formula the *non-overlapping discretization* is obtained. This is another discrete formulation that is equivalent to the *original-problem*, except that it constitutes a *non-overlapping discretization*; and

5. Thereafter, each one of the *coarse-mesh* subdomains is assigned to a different processor and the code is programmed separately in each one of the processors.

The theoretical DVS-framework is very elegant; in it, the algebraic operations can be carried out systematically and with great simplicity. Furthermore, many simplifying algebraic results have been obtained in previous work (Herrera *et al*., 2014 and Herrera and Yates, 2011). To optimize the communications and processing time a purely algebraic critical-route is defined, which profits much from such algebraic results previously obtained. Then, this algebraic critical-route is transformed into a computational code using C++ and several well-established computational techniques such as MPI.

Following the steps indicated above, in the present paper software for problems of isotropic elastic solids in equilibrium has been developed and tested experimentally. The high parallelization efficiency of the software so obtained has been verified experimentally. To be specific, only the DVS-BDDC algorithm has been implemented for this problem. However, by simple combinations of the routines already developed the other *DVS-algorithms* can be implemented.

The standard discretization

Following the steps succinctly described in overview of DVS-software, software that constitutes a tool for effectively applying massively parallel hardware to isotropic elastic solids in equilibrium was constructed. In particular, it permits to treat the following boundary value problem (BVP):

$$(\lambda+\mu)\,\nabla\nabla\bullet\underline{u}+\mu\Delta\underline{u}=\underline{f}_\Omega \qquad (3.1)$$

Subjected to the *Dirichlet* boundary conditions:

$$\underline{u}=0,\ \text{on}\ \partial \qquad (3.2)$$

By simple modifications of the code, other boundary conditions can also be accommodated.

The software that we have developed treats in parallel the discrete system of linear equations that is obtained when the *standard discretization method* used to obtain the *original discretization* of the Dirichlet BVP defined by Eqs. (3.1) and (3.2) is the finite element method (FEM). In particular, it was obtained applying the well-known variational principle:

$$\int_\Omega \left\{(\lambda+\mu)(\nabla\bullet\underline{u})(\nabla\bullet\underline{w})+\mu\nabla\underline{u}:\nabla\underline{w}\right\}dx= \int_\Omega \underline{f}_\Omega\bullet\underline{w}dx \qquad (3.3)$$

with linear functions.

Such system of equations can be written as

$$\underline{\underline{M}}\,\underline{U}=\underline{F} \qquad (3.4)$$

Here, it is understood that the vectors $\underline{U}$ and $\underline{F}$, are functions defined on the whole set of *original-nodes* of the mesh used in the FEM discretization, whose values at each node are 3-D vectors. They can be written as $\underline{U}\equiv(\underline{U}_p)\equiv(U_{pi})$ and $\underline{F}\equiv(\underline{F}_p)\equiv(F_{pi})$. As for the matrix $\underline{\underline{M}}$, the notation

$$\underline{\underline{M}}\equiv(\underline{M}_{pq})\equiv(M_{piqj}) \qquad (3.5)$$

is adopted. Above, the range of p *and* q is the whole set of *original-nodes*, while i and j may take any of the values 1, 2, 3.

Preliminary notions and notations

The *DVS-approach* is based on *non-overlapping discretizations*, which were introduced during its development (Herrera *et al*., 2014). A discretization is *non-overlapping* when it is based on a system of nodes that is *non-overlapping*; to distinguish the nodes of such a system from the *original-nodes*, they are called *derived-nodes*. In turn, a system of nodes is *non-overlapping*, with respect to a *coarse-mesh* (or, domain-decomposition), if each one of them belongs to one and only one of the domain-decomposition subdomains. In the general DVS-framework, the *derived-vector space* (DVS) is constituted by the whole linear space of functions whose domain is the total set of *derived-nodes* and take values in $\mathbb{R}^n$. In the present paper, where problems of elasticity that are governed by a system of three PDEs are treated, we take $n = 3$. Usually, when the basic mathematical model is governed by a single differential equation, n is chosen to be equal to 1.

Generally, when the *coarse-mesh* is introduced some of the nodes of the *fine-mesh* fall in the closures of more than one subdomain of the *coarse-mesh*. When that is the case, a general procedure for transforming such an overlapping set of nodes into a non-overlapping one was introduced in previous papers (see Herrera *et al*., 2014). Such a procedure consists in dividing each *original-node* into as many pieces as subdomains it belongs to, and then allocating one and only one of such pieces to each one of the subdomains. For a case in which the *coarse-mesh* consists of only four subdomains, this process is schematically illustrated in Figures 1 to 4.

Then, the final result is the system of *non-overlapping nodes* shown in Figure 4. Each one of the *non-overlapping* nodes is uniquely identified by the pair (p, α), where p is the *original-node* it comes from and α is the subdomain it belongs to. Using this notation, for each fixed $\beta = 1, ..., E$, it is useful to define $X^\beta \subset X$ as follows: The derived node (p, α) belongs to X^β, if and only if, $\alpha = \beta$.

In what follows, the family of subsets$\{X^1, ..., X^E\}$ just defined will be referred to as the *non-overlapping decomposition of the set of derived-nodes*. This because this family of subsets of X possesses the following property:

$$X = \bigcup_{\alpha=1}^{E} X^\alpha \; and \; \varnothing = X^\alpha \cap X^\beta \; when \; \alpha \neq \beta, \tag{4.1}$$

An important property implied by Eq. (4.1) is that the *derived-vector space*, W, is the direct-sum of the following family of subspaces of W : $\{W^1, ..., W^E\}$; i.e.,

$$W = W^1 \oplus ... \oplus W^E \tag{4.2}$$

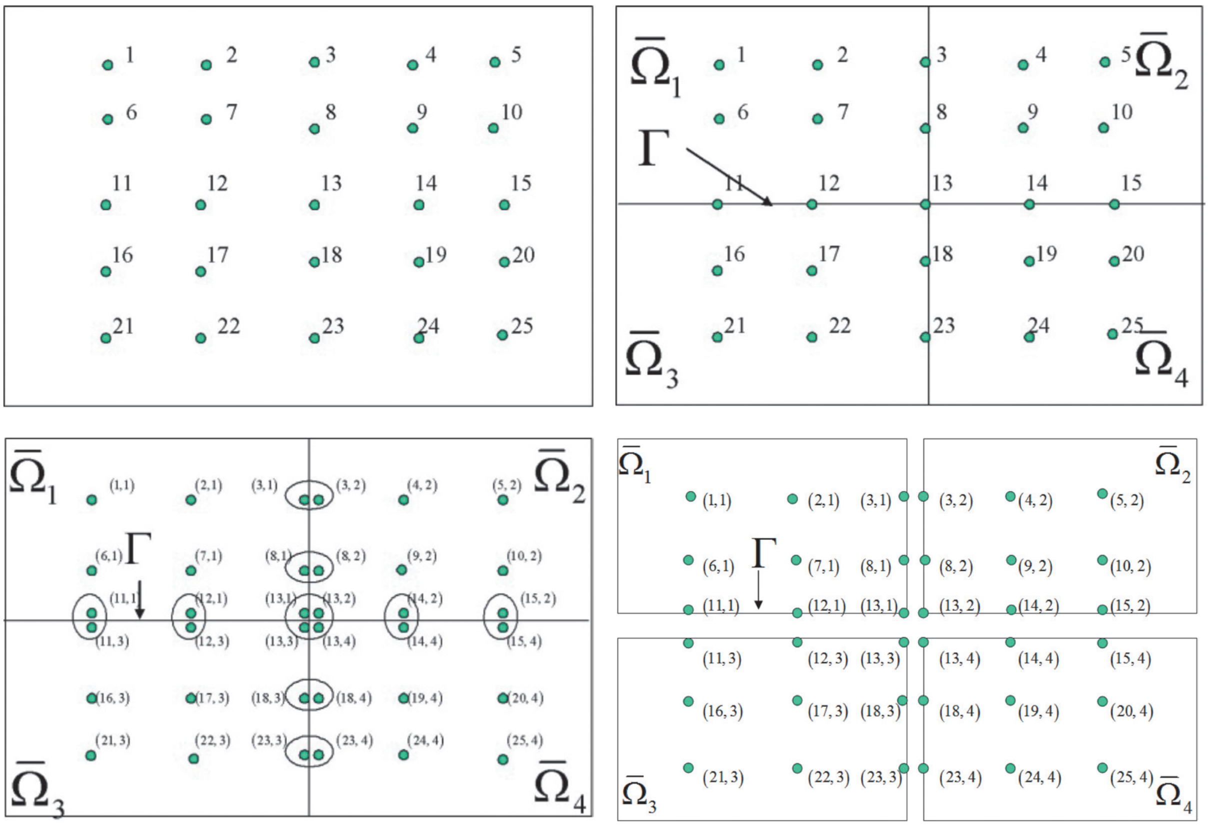
Here, we have written

$$W^\alpha \equiv W(X^\alpha), \alpha = 1, ..., E \tag{4.3}$$

The notation $W(X^\alpha)$, introduced previously (see, for example Herrera *et al.*, 2014), is here used to represent the linear subspace of W whose vectors vanish at every *derived-node* that does not belong to X^α. An important implication, very useful for developing codes in parallel, is that every *derived-vector* $\underline{w} \in W$ can be written uniquely in the form

$$\underline{w} = \sum_{\alpha=1}^{E} \underline{w}^\alpha, \; with \; \underline{w}^\alpha \in W^\alpha \tag{4.4}$$

As it is customary in DDM developments, in the *DVS-approach* a classification of the



Figures 1 to 4.

nodes used is introduced. We list next the most relevant subsets of X used in what follows:

I *internal nodes*
Γ *interface nodes*
π *primal nodes*
Δ *dual nodes*
$\Pi \equiv \mathrm{I} \cup \pi$ *'extended primal' nodes*
$\Sigma \equiv \mathrm{I} \cup \Delta$ *'extended dual' nodes*

Also, we observe that each one of the following set-families are disjoint: $\{\mathrm{I}, \Gamma\}$, $\{\mathrm{I}, \pi, \Delta\}$, $\{\Pi, \Delta\}$ and $\{\Sigma, \pi\}$, while

$$X = I \cup \Gamma = I \cup \pi \cup \Delta = \Pi \cup \Delta = \Sigma \cup \pi \tag{4.6}$$

Next, we highlight some of most important notation and nomenclature used in the *DVS-framework*; for further details the reader is referred to previous works of this line of research (in particular (Herrera *et al*., 2014), where additional references are given). When considering any given *derived-node*, which is identified by the pair of numbers (p, α), the natural number p (which corresponds to an *original-node*) is called the *'ancestor'* of the *derived-node*, while α (which ranges from 1 to E) identifies the subdomain it belongs to. Furthermore, for every *original-node* $p \in \widehat{X}$, the notation $\mathrm{Z}(p) \subset \mathrm{X}$ will be used to represent the set of *derived-nodes* that derived from it. Then, the *'multiplicity of p'*, $m(p)$, is the *cardinality* of $\mathrm{Z}(p)$.

We observe that the *multiplicity of p* is defined as a property of each *original-node*, p. There is another kind of multiplicity that is used in the *DVS-framework*, which is defined as a property of each pair (p, q) of *original-nodes* and is also used in the *DVS-framework* theory. To introduce it, we define

$$\delta_{pq}^{\alpha} \equiv \begin{matrix} 1, if \;\; p,q \in \overline{}_{\alpha} \\ 0, otherwise \end{matrix}, \alpha = 1,...,E; and \tag{4.7}$$

Then, the multiplicity of the pair (p, q) -written as $m(p, q)$- is defined to be

$$m(p,q) \equiv \sum_{\alpha=1}^{E} \delta_{pq}^{\alpha} \tag{4.8}$$

When $\underline{u}$ is a *derived-vector*, so that $\underline{u}$ is a function defined on X, $\underline{u}(p, \alpha)$ stands for the value of $\underline{u}$ at the *derived-node* (p, α). In particular, in applications of the *DVS-framework* to elasticity problems, those values are -vectors and the real-number $u(p, \alpha, i)$ - $i = 1, 2, 3$- will be the *i–th* component of the vector $\underline{u}(p, \alpha)$. The *derived-vector space* is supplied with an inner product, the *'Euclidean inner-product'*, which using the above notation for every pair of *derived-vectors*, $\underline{u}$ and $\underline{w}$, is defined by

$$\underline{u} \bullet \underline{w} \equiv \sum_{(p,\alpha)\in \mathrm{X}} \sum_{i=1}^{3} u(p,\alpha,i)\, w(p,\alpha,i) = \sum_{\alpha=1}^{E} \sum_{(p,\alpha)\in \mathrm{X}^{\alpha}} \sum_{i=1}^{3} u(p,\alpha,i)\, w(p,\alpha,i) \tag{4.9}$$

For the parallelization of the algorithms the relation

$$\underline{u} \bullet \underline{w} = \sum_{\alpha=1}^{E} \underline{u}^{\alpha} \bullet \underline{w}^{\alpha}, \; whenever \; \underline{u}^{\alpha}, \underline{w}^{\alpha} \in W^{\alpha} \tag{4.10}$$

will be useful, because the vector-components corresponding to different subdomains will be handled by different processors when implementing them.

Let $p \in \widehat{X}$, be an *original-node* and $\underline{u} \in W$ a *derived-vector*. Then, $\underline{u} \in W$ is said to be *'continuous at p'* when $\underline{u}(p, \alpha)$ is independent of α, and it is said to be of *'zero-average at p'* when

$$\sum_{\alpha \in Z(p)} \underline{u}(p,\alpha) = 0 \tag{4.11}$$

When the corresponding properties are satisfied for every $p \in \widehat{X}$, the derived-vector $\underline{u}$ is simply said to be *'continuous'* or *'zero-average'*. The linear subspaces W_{12} and W_{11} of W, are constituted by the *continuous* and the *zero-average* vectors of W, respectively. These two subspaces are orthogonal complements of each other. The matrices $\underline{\underline{a}}$ and $\underline{\underline{j}}$ are the orthogonal projections on W_{12} and on W_{11}, respectively. They satisfy:

$$\underline{\underline{a}} + \underline{\underline{j}} = \underline{\underline{I}} \tag{4.12}$$

where $\underline{\underline{I}}$ is the identity matrix. For any $\underline{w} \in W$, the explicit evaluation of $\underline{v} \equiv \underline{\underline{a}}\underline{w}$ is given by:

$$\underline{v}(p,\alpha) \equiv \frac{1}{m(p)} \sum_{(p,\beta)\in Z(p)} \underline{w}(p,\beta) \quad (4.13)$$

Using this equation, the evaluation of $\underline{\underline{j}}\underline{w}$ is also straight forward, since Eq. (4.12) implies

$$\underline{\underline{j}}\underline{w} = \underline{w} - \underline{\underline{a}}\underline{w} \quad (4.14)$$

The *natural injection* of $\widehat{W}$ into W, written as $R{:}\widehat{W} \to W$, is defined for every $\widehat{\underline{u}} \in \widehat{W}$ by

$$\left(R\widehat{\underline{u}}\right)(p,\alpha) = \widehat{\underline{u}}(p),\ \forall (p,\alpha) \in X \quad (4.15)$$

When $\widehat{\underline{u}} \in \widehat{W}$, $\left(R\widehat{\underline{u}}\right) \in W_{12}$ necessarily. We observe that $R\widehat{W} = W_{12}$. Furthermore, it can be seen that R has a unique inverse in W_{12}; i.e., $R^{-1}: W_{12} \to \widehat{W}$ is well-defined.

How to build non-overlapping discretizations

This Section explains how to transform a standard (*overlapping*) discretization into a *non-overlapping discretization*. The DVS procedure here explained permits transforming an *overlapping discretization* into a *non-overlapping discretizations* and yields directly *preconditioned* algorithms that are subjected to *constraints*. It can be applied whenever the following basic assumption is fulfilled:

$$m(p,q) = 0 \Rightarrow M_{pq} = 0 \quad (5.1)$$

Here, the symbol $\Rightarrow$ stands for the logical implication and it is understood that $\underline{\underline{M}}$ is the matrix occurring in Eq. (2.1).

We define the matrix $\underline{\underline{a}}'$ by its action on any vector of W: when $\underline{u} \in W$, we have

$$\underline{\underline{a}}'\underline{u} = \underline{u}_I + \underline{u}_\Delta + \underline{\underline{a}}\underline{u}_\pi \quad (5.2)$$

We observe that the action of $\underline{\underline{a}}$ can be carried out by applying the operator at the *primal-nodes* exclusively. Then, we define the *'constrained space'* by

$$W^r \equiv \underline{\underline{a}}'W \quad (5.3)$$

Clearly, $W^r \subset W$ is a linear subspace of W and for any $\underline{u} \in W$, $\underline{\underline{a}}'\underline{u}$ is the projection of $\underline{u}$, on W^r.

Now, we define

$$s(p,q) \equiv \begin{cases} 1, \textit{when}\ m(p,q) = 0 \\ m(p,q), \textit{when}\ m(p,q) \neq 0 \end{cases} \quad (5.4)$$

For $\gamma = 1, ..., E$, we define the matrices

$$\underline{\underline{M}}^\gamma \equiv \left(M^\gamma_{pq}\right) \textit{with}\ M^\gamma_{pq} \equiv \frac{M_{pq}}{s(p,q)} \delta^\gamma_{pq} \quad (5.5)$$

Next, we define the matrices:

$$\underline{\underline{A}}^\gamma \equiv \left(A^\gamma_{(p,\alpha)(q,\beta)}\right) \textit{with}\ A^\gamma_{(p,\alpha)(q,\beta)} \equiv M^\gamma_{pq}\delta_{(\alpha,\gamma)}\delta_{(\beta,\gamma)} \quad (5.6)$$

and

$$\underline{\underline{A}}^t \equiv \sum_{\gamma=1}^{E} \underline{\underline{A}}^\gamma \quad (5.7)$$

Then, we define

$$\underline{\underline{A}} \equiv \underline{\underline{a}}'\underline{\underline{A}}^t\underline{\underline{a}}' \quad (5.8)$$

The following result was shown in previous papers (Herrera *et al.*, 2014):

Theorem 5.1.- Let $\underline{U} \in \widehat{W}$ and $\underline{u} \in W$ be related by $\underline{u} = R\underline{U}$, while $\underline{f} \in W_{12}$ is defined by

$$\underline{f} \equiv R\left(\widehat{\underline{\underline{m}}}^{-1}\underline{F}\right) \quad (5.9)$$

Here, $\widehat{\underline{\underline{m}}}$ is a diagonal matrix that transforms $\widehat{W}$ into itself, whose diagonal-values are $m(p)$, while here its inverse is denoted by $\widehat{\underline{\underline{m}}}^{-1}$. Then, the discretized version of static elasticity of Eq.: (3.4):

$$\underline{\underline{M}}\underline{U} = \underline{F} \quad (5.10)$$

is fulfilled, if and only if

$$\underline{\underline{a}}\underline{\underline{A}}\underline{u} = \underline{f}\ \textit{and}\ \underline{\underline{j}}\underline{u} = 0 \quad (5.11)$$

Proof.- See for example (Herrera *et al*., 2014).

The preconditioned DVS-algorithms with constraints

There are four *DVS-algorithms* (Herrera *et al*., 2014), and two of them are the DVS-BDDC and the DVS- FETI-DP. These are DVS versions of the well-known BDDC (Dohrmann, 2003), (Mandel *et al*., 2005) and FETI-DP (Farhat and Roux, 1991), (Farhat *et al*., 2000). As for the other two, nothing similar had been reported in the literature prior to the publication of the *DVS-algorithms*. By now, it is well known that BDDC and FETI-DP are closely related and the same can be said of the whole group of four DVS-algorithms.

The DVS-Schur-complement is defined by

$$\underline{\underline{S}} \equiv \underline{\underline{A}}_{\Delta\Delta} - \underline{\underline{A}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{\underline{A}}_{\Pi\Delta} \qquad (6.1)$$

We also define

$$\overline{\underline{f}}_{\Delta} \equiv \underline{f}_{\Delta} - \underline{\underline{A}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{f}_{\Pi} \qquad (6.2)$$

Then, writing $\underline{u} \equiv \underline{u}_{\Pi} + \underline{u}_{\Delta}$ it has been shown (Herrera *et al*., 2014) that Eq. (5.11) is fulfilled if and only if

$$\underline{\underline{a}}\underline{\underline{S}}\underline{u}_{\Delta} = \overline{\underline{f}}_{\Delta}, \quad \underline{\underline{j}}\underline{u}_{\Delta} = 0 \qquad (6.3)$$

and

$$\underline{u}_{\Pi} = \left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\left(\underline{f}_{\Pi\Delta} - \underline{\underline{A}}_{\Pi\Delta}\underline{u}_{\Delta}\right) \qquad (6.4)$$

The general strategy followed in the DVS approach, is to find $\underline{u}_{\Delta} \in W(\Delta)$ first and then apply Eq. (6.4) to obtain the remaining part, $\underline{u}_{\Pi} \in W(\Pi)$, of $\underline{u}$. For this strategy to be effective it is essential that the application of $\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}$ be computationally cheap. Different DVS-algorithms are derived by seeking different pieces of information such that $\underline{u}_{\Delta} \in W(\Delta)$ can be derived from it in a computationally-cheap manner. In particular, the four DVS-algorithms mentioned before seek for: $\underline{u}_{\Delta}$, $\underline{\underline{j}}\underline{\underline{S}}\underline{u}_{\Delta}$, $\underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{\underline{S}}\underline{u}_{\Delta}$ and $\underline{\underline{S}}\underline{u}_{\Delta}$, respectively. Drawing from (Herrera *et al*., 2014), they are here listed.

The DVS-BDDC algorithm

This algorithm seeks for $\underline{u}_{\Delta}$. It is:

$$\underline{\underline{a}}\underline{\underline{S}}^{-1}\underline{\underline{a}}\underline{\underline{S}}\underline{u}_{\Delta} = \underline{\underline{a}}\underline{\underline{S}}^{-1}\underline{f}_{\Delta} \text{ and } \underline{\underline{j}}\underline{u}_{\Delta} = 0 \qquad (6.5)$$

The DVS-primal-algorithm

We set $\underline{v}_{\Delta} \equiv \underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{\underline{S}}\underline{u}_{\Delta}$ and the algorithm consists in searching for a function $\underline{v}_{\Delta} \in W_{\Delta}$, which fulfills:

$$\underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{\underline{S}}\underline{\underline{j}}\underline{v}_{\Delta} = \underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{f}_{\Delta} \text{ and } \underline{\underline{a}}\underline{\underline{S}}\underline{v}_{\Delta} = 0 \qquad (6.6)$$

Once $\underline{v}_{\Delta} \in W(\Delta)$ has been obtained, then

$$\underline{u}_{\Delta} = \underline{\underline{a}}\left(\underline{\underline{S}}^{-1}\underline{f}_{\Delta} + \underline{v}_{\Delta}\right) \qquad (6.7)$$

The DVS-feti-dp algorithm

This algorithm seeks for $\underline{\lambda} \equiv \underline{\underline{j}}\underline{\underline{S}}\underline{u}_{\Delta}$. Thus, the algorithm is: "Given $\underline{f}_{\Delta} \in \underline{\underline{a}}W_{\Delta}$, find $\underline{\lambda}_{\Delta} \in W_{\Delta}$ such that

$$\underline{\underline{j}}\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{\lambda} = -\underline{\underline{j}}\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{f}_{\Delta} \text{ and } \underline{\underline{a}}\underline{\lambda} = 0 \qquad (6.8)$$

Once $\underline{\lambda} \in W_{\Delta}$ has been obtained, $\underline{u}_{\Delta} \in \underline{\underline{a}}\, W_{\Delta}$ is given by:

$$\underline{u}_{\Delta} = \underline{\underline{a}}\underline{\underline{S}}^{-1}\left(\underline{f}_{\Delta} + \underline{\lambda}\right) \qquad (6.9)$$

The DVS-dual-algorithm

In this case one seeks for $\underline{\mu} \equiv \underline{\underline{S}}\underline{u}_{\Delta}$ using the relation:

$$\underline{\underline{S}}\underline{\underline{a}}\underline{\underline{S}}^{-1}\underline{\underline{a}}\underline{\mu} = \underline{\underline{S}}\underline{\underline{a}}\underline{\underline{S}}^{-1}\underline{f}_{\Delta} \text{ and } \underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{\mu} = 0 \qquad (6.10)$$

Once $\underline{\mu}_{\Delta} \in W(\Delta)$ has been obtained, $\underline{u}_{\Delta} \in W(\Delta)$ is given by:

$$\underline{u}_{\Delta} = \underline{\underline{S}}^{-1}\underline{\mu} \qquad (6.11)$$

The elementary pieces of DVS-software

All the DVS-algorithms are iterative algorithms and can be implemented with recourse to Conjugate Gradient Method (CGM), when the matrix is definite and symmetric, as is the case of elasticity problems here considered, or some other iterative procedure such as GMRES, when that is not the case. At each iteration step, depending on the *DVS-algorithm* that is applied, one has to compute the action on an arbitrary *derived-vector* of one of the following matrices:

$\underline{\underline{a}}\underline{\underline{S}}^{-1}\underline{\underline{a}}\underline{\underline{S}}$, $\underline{\underline{j}}\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}$, $\underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{\underline{S}}\underline{\underline{j}}$ or $\underline{\underline{S}}\underline{\underline{a}}\underline{\underline{S}}^{-1}\underline{\underline{a}}$. In turn, such matrices are different permutations of $\underline{\underline{S}}$, $\underline{\underline{S}}^{-1}$, $\underline{\underline{a}}$ and $\underline{\underline{j}}$. Thus, a code for implementing any of the DVS-algorithms can be easily developed when codes for carrying out the action of each one of such matrices are already available.

To produce such codes will be the goal of the next Section, while the remaining of this one is devoted to obtain some auxiliary results that will be used there and were previously presented in (Herrera *et al*., 2014). The first one of such results is:

$$\underline{\underline{S}} \equiv \underline{\underline{A}}^t_{\Delta\Delta} - \underline{\underline{A}}^t_{\Delta\Pi}\left(\underline{\underline{a}}'\underline{\underline{A}}^t_{\Pi\Pi}\underline{\underline{a}}'\right)^{-1}\underline{\underline{a}}'\underline{\underline{A}}^t_{\Pi\Delta} \qquad (7.1)$$

The second one is: When $\underline{w}\in W$, the following identity holds

$$\underline{\underline{S}}^{-1}\underline{w} = \left(\underline{\underline{A}}^{-1}\underline{w}_\Delta\right)_\Delta \qquad (7.2)$$

Here the notation $\left(\underline{\underline{A}}^{-1}\underline{w}_\Delta\right)_\Delta$ stands for the component on $W(\Delta)$ of $\underline{\underline{A}}^{-1}\underline{w}_\Delta$.

The third and fourth results required refer to the *pseudo-inverses* that occur in Eqs. (7.1) and (7.2). They are:

Let $\underline{w}\in W^r(\Pi)$ and $\underline{v} \equiv \left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{w}$, then

$$\underline{\underline{a}}'\left(\underline{\underline{A}}^t_{\pi\pi} - \underline{\underline{A}}^t_{\pi I}\left(\underline{\underline{A}}^t_{II}\right)^{-1}\underline{\underline{A}}^t_{I\pi}\right)^{-1}\underline{v}_\pi =$$
$$\underline{w}_\pi - \underline{\underline{A}}^t_{\pi I}\left(\underline{\underline{A}}^t_{II}\right)^{-1}\underline{w}_I,\ and\ \underline{\underline{j}}\underline{v}_\pi = 0 \qquad (7.3)$$

together with

$$\underline{v}_I = \left(\underline{\underline{A}}^t_{II}\right)^{-1}\left(\underline{w}_I - \underline{\underline{A}}^t_{I\pi}\underline{v}_\pi\right) \qquad (7.4)$$

Let $\underline{w}\in W^r$ and $\underline{v} \equiv \underline{\underline{A}}^{-1}\underline{w}$, then

$$\underline{\underline{a}}'\left(\underline{\underline{A}}^t_{\pi\pi} - \underline{\underline{A}}^t_{\pi\Sigma}\left(\underline{\underline{A}}^t_{\Sigma\Sigma}\right)^{-1}\underline{\underline{A}}^t_{\Sigma\pi}\right)\underline{v}_\pi =$$
$$\underline{w}_\pi - \underline{\underline{A}}^t_{\pi\Sigma}\left(\underline{\underline{A}}^t_{\Sigma\Sigma}\right)^{-1}\underline{w}_\Sigma,\ and\ \underline{\underline{j}}\underline{v}_\pi = 0 \qquad (7.5)$$

together with

$$\underline{v}_\Sigma = \left(\underline{\underline{A}}^t_{\Sigma\Sigma}\right)^{-1}\left(\underline{w}_\Sigma - \underline{\underline{A}}^t_{\Sigma\pi}\underline{v}_\pi\right) \qquad (7.6)$$

These two results permit applying iterative algorithms, in which the CGM is used, when the actions of $\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}$ and $\underline{\underline{A}}^{-1}$, respectively, are computed.

Construction of the DVS-software

All the DVS-algorithms presented in the Section on the preconditioned DVS-algorithms with constraints are iterative, as is the case with most DDM algorithms, and to implement them it is only necessary to develop parallelized codes capable of computing the action of each one of the matrices $\underline{\underline{S}}$, $\underline{\underline{S}}^{-1}$, $\underline{\underline{a}}$ or $\underline{\underline{j}}$ on an arbitrary *derived-vector*, as it was foreseen in (Herrera *et al*., 2014).

In the code here reported, all system-of-equations' solutions that were non-local were obtained with the help of the CGM algorithm. Due to this fact, actually the following subprograms were required: $\underline{\underline{S}}$, $\underline{\underline{S}}^{-1}$ and $\left(\underline{\underline{a}}\underline{\underline{S}}^{-1}\underline{\underline{a}}\ \underline{\underline{S}}\right)^{-1}$. Furthermore, the application of

$$\underline{\underline{S}} = \underline{\underline{A}}^t_{\Delta\Delta} - \underline{\underline{A}}^t_{\Delta\Pi}\left(\underline{\underline{a}}'\underline{\underline{A}}^t_{\Pi\Pi}\underline{\underline{a}}'\right)^{-1}\underline{\underline{a}}'\underline{\underline{A}}^t_{\Pi\Delta} \qquad (8.1)$$

requires to compute the action of $\left(\underline{\underline{a}}'\underline{\underline{A}}^t_{\Pi\Pi}\underline{\underline{a}}'\right)^{-1}$ which is non-local. Thus, an efficient subprogram to carry-out this operation efficiently in parallel was required and was developed.

The communications required by *DVS-algorithms* are very easy to analyze. Indeed, when a different processor is allocated to each one of the coarse-mesh subdomains (i.e., to the subsets of *derived-nodes*, X^α, $\alpha = 1, ..., E$, of the *non-overlapping partition of* X) –as it was done in the work here reported– transmission of information between different processors occurs only when the *global Euclidean inner-product* is computed, or either the matrix $\underline{\underline{a}}$ or the matrix $\underline{\underline{a}}'$ is applied. Furthermore, in these operations the amount of information transmitted is very small.

In a first tentative version of the software, a *master-processor* was also used. However, using such a *master-processor* as a communications center is very time-costly and when the *master-processor* is not used as a communications center the work done by it is so small that it can be eliminated easily. When this is done, the performance of the DVS-algorithm became extremely good as it is explained and discussed in the Section on Numerical Results.

Only the DVS-BDDC algorithm was implemented. Although the implementation of the other three DVS-algorithms is very similar, and their expected parallel efficiency as well, their implementation would have taken additional time and effort that we preferred to save for future work.

Construction of the local DVS-software

A fundamental property of $\underline{\underline{A}}^t$, as defined by Eq. (5.7) is that it is block-diagonal, in which each one of the blocks is $\underline{\underline{A}}^\alpha$, for each $\alpha = 1, ..., E$, is a linear-transformation of W^α into itself. This property simplifies very much the parallelization of the codes to implement the DVS-algorithms.

To this end, each one of the subsets X^α of the *non-overlapping decomposition of* X, is assigned to a different processor and the set of processors is numbered accordingly. The fact that every vector $\underline{w} \in W$ can be written in a unique manner as

$$\underline{w} = \sum_{\alpha=1}^{E} \underline{w}_\alpha, \; with \; \underline{w}_\alpha \in W^\alpha \tag{9.1}$$

is used for this purpose. The processor γ handles only the $\underline{w}_\gamma$ component of every vector $\underline{w} \in W$. Then, all the operations of the processor γ transform $\underline{w}_\gamma$ into a vector that also belongs to W^γ; even the operators $\underline{\underline{a}}$ and $\underline{\underline{a}}'$ transform $\underline{w}_\gamma$ into a vector of W^γ, except that $\underline{\underline{a}}$ and $\underline{\underline{a}}'$ require information from of a few neighboring processors. However, it is important to make sure that such information be updated at the time it is gathered.

When evaluating the action on a vector of any of the matrices considered, processor γ will be responsible of constructing the γ component of such a vector; in particular, $\left(\underline{\underline{S}}\underline{w}\right)_\gamma$, $\left(\underline{\underline{S}}^{-1}\underline{w}\right)_\gamma$, $\left(\underline{\underline{a}}\underline{w}\right)_\gamma$, or $\left(\underline{\underline{j}}\underline{w}\right)_\gamma$, depending on the matrix that is being applied. In what follows it is assumed that, from the start, the nodes of the set X^γ have been classified into I: *internal,* π: *primal,* and Δ: *dual*. Other *node-classes* of X^γ that will be considered are: Π: *extended-primal,* and Σ: *extended-dual*. Without any further notice, the following relation will also be used:

$$W^\gamma \equiv W^\gamma(I) \oplus W^\gamma(\pi) \oplus W^\gamma(\Delta) = W^\gamma(\Pi) \oplus W^\gamma(\Delta) = W^\gamma(\Sigma) \oplus W^\gamma(\pi) \tag{9.2}$$

The application of $\underline{\underline{a}}$, $\underline{\underline{a}}'$ *and* $\underline{\underline{j}}$

To start with, we evaluate $\left(\underline{\underline{a}}\underline{w}\right)_\gamma$ when $\underline{w} \in W^\gamma$. As it will be seen, the application of $\underline{\underline{a}}$ to any vector of W^γ requires exchange of information between processor γ and other processors. Indeed, recalling Eq. (4.13) we have

$$\left(\underline{\underline{a}}\underline{w}\right)_\gamma = \underline{\underline{a}}\underline{w}(p,\gamma) = \frac{1}{m(p)} \sum_{(p,\beta)\in Z(p)} \underline{w}(p,\beta) \tag{9.3}$$

Thus, this operation requires information from the processors that possess *derived-nodes* belonging to $Z(p)$; therefore, its computation involves communications between different processors, which may slow the processing. In view of Eq. (9.3), it is clear that except for this exchange of information, the evaluation of $\left(\underline{\underline{a}}\underline{w}\right)_\gamma$, is very simple. Once $\underline{\underline{a}}\underline{w}$,has been obtained, the relation $\underline{\underline{j}}\underline{w} = \underline{w} - \underline{\underline{a}}\underline{w}$,can be used to compute the action of $\underline{\underline{j}}$. As for the action of $\underline{\underline{a}}'$, we recall that $\underline{\underline{a}}'$ is obtained when the application of $\underline{\underline{a}}$ is restricted to *primal-nodes*.

Before going ahead, some final comments are in order. The application of $\underline{\underline{a}}$, and hence that of $\underline{\underline{a}}'$, also requires transmission of information between the processors. Thus, for enhancing the efficiency of the codes it is essential that the application procedures be designed with great care. As it will be seen, with a few exceptions, all the exchange of information required when the DVS-algorithms are implemented is when the transformations $\underline{\underline{a}}$ and $\underline{\underline{a}}'$ are applied.

The DVS-software for $\underline{\underline{S}}$ *and* $\underline{\underline{S}}^{-1}$

It should be observed that in view of the definition of the matrix $\underline{\underline{A}}^\gamma$ and the submatrices occurring in the following decomposition:

$$\underline{\underline{A}}^\gamma = \begin{pmatrix} \underline{\underline{A}}^\gamma_{II} \; \underline{\underline{A}}^\gamma_{I\pi} \; \underline{\underline{A}}^\gamma_{I\Delta} \\ \underline{\underline{A}}^\gamma_{\pi I} \; \underline{\underline{A}}^\gamma_{\pi\pi} \; \underline{\underline{A}}^\gamma_{\pi\Delta} \\ \underline{\underline{A}}^\gamma_{\Delta I} \; \underline{\underline{A}}^\gamma_{\Delta\pi} \; \underline{\underline{A}}^\gamma_{\pi\Delta} \end{pmatrix} \tag{9.4}$$

for any they transform vectors of $W(X^\gamma)$ into vectors of $W(X^\gamma)$. Therefore, the *local* matrix $\underline{\underline{Q}}$ is defined to be

$$\underline{\underline{Q}} \equiv \underline{\underline{A}}^\gamma \tag{9.5}$$

where $\underline{\underline{A}}^\gamma$ is the matrix defined in how to build non-overlapping discretizations, by Eq. (5.6). In this equation the index γ is omitted in the definition of $\underline{\underline{Q}}$, because γ is kept fixed. Due to the comments already made, it is clear that $\underline{\underline{Q}}$ is a well-defined linear transformation of $\overline{\overline{W}}^\gamma$ into itself. In particular, when $\underline{w}^\gamma \in W^\gamma$, the computation of $\left(\underline{\underline{Q}}\,\underline{w}^\gamma\right)_\gamma$ can be carried out in an autonomous manner, at processor γ, without exchange of information with other processors. This is a fundamental difference with $\underline{\underline{a}}$, $\underline{\underline{a}}'$ and $\underline{\underline{j}}$, and implies that at each processor either the matrix $\underline{\underline{Q}}$ is constructed, or internal software capable of evaluating its action on any vector of W^γ is made available.

In view of Eq. (9.4), the matrix $\underline{\underline{Q}}$ will be written in two forms

$$\underline{\underline{Q}} = \begin{pmatrix} \begin{pmatrix} \underline{\underline{Q}}_{II} & \underline{\underline{Q}}_{I\pi} \\ \underline{\underline{Q}}_{\pi I} & \underline{\underline{Q}}_{\pi\pi} \end{pmatrix} & \begin{pmatrix} \underline{\underline{Q}}_{I\Delta} \\ \underline{\underline{Q}}_{\pi\Delta} \end{pmatrix} \\ \begin{pmatrix} \underline{\underline{Q}}_{\Delta I} & \underline{\underline{Q}}_{\Delta\pi} \end{pmatrix} & \begin{pmatrix} \underline{\underline{Q}}_{\Delta\Delta} \end{pmatrix} \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} \underline{\underline{Q}}_{II} & \underline{\underline{Q}}_{I\Delta} \\ \underline{\underline{Q}}_{\Delta I} & \underline{\underline{Q}}_{\Delta\Delta} \end{pmatrix} & \begin{pmatrix} \underline{\underline{Q}}_{I\pi} \\ \underline{\underline{Q}}_{\Delta\pi} \end{pmatrix} \\ \begin{pmatrix} \underline{\underline{Q}}_{\pi I} & \underline{\underline{Q}}_{\pi\Delta} \end{pmatrix} & \begin{pmatrix} \underline{\underline{Q}}_{\pi\pi} \end{pmatrix} \end{pmatrix} \tag{9.6}$$

The following expressions, which are clear in view of Eq. (9.6), will be used in the sequel:

$$\underline{\underline{Q}} = \begin{pmatrix} \underline{\underline{Q}}_{\Pi\Pi} & \underline{\underline{Q}}_{\Pi\Delta} \\ \underline{\underline{Q}}_{\Delta\Pi} & \underline{\underline{Q}}_{\Pi\Delta} \end{pmatrix} = \begin{pmatrix} \underline{\underline{Q}}_{\Sigma\Sigma} & \underline{\underline{Q}}_{\Sigma\pi} \\ \underline{\underline{Q}}_{\pi\Sigma} & \underline{\underline{Q}}_{\pi\pi} \end{pmatrix} \tag{9.7}$$

Here:

$$\underline{\underline{Q}}_{\Pi\Pi} \equiv \begin{pmatrix} \underline{\underline{Q}}_{II} & \underline{\underline{Q}}_{I\pi} \\ \underline{\underline{Q}}_{\pi I} & \underline{\underline{Q}}_{\pi\pi} \end{pmatrix}, \underline{\underline{Q}}_{\Pi\Delta} \equiv \begin{pmatrix} \underline{\underline{Q}}_{I\Delta} \\ \underline{\underline{Q}}_{\pi\Delta} \end{pmatrix}$$
$$\underline{\underline{Q}}_{\Delta\Pi} \equiv \begin{pmatrix} \underline{\underline{Q}}_{\Delta I} & \underline{\underline{Q}}_{\Delta\pi} \end{pmatrix}, \underline{\underline{Q}}_{\Delta\Delta} \equiv \begin{pmatrix} \underline{\underline{Q}}_{\Delta\Delta} \end{pmatrix} \tag{9.8}$$

and

$$\underline{\underline{Q}}_{\Sigma\Sigma} \equiv \begin{pmatrix} \underline{\underline{Q}}_{II} & \underline{\underline{Q}}_{I\Delta} \\ \underline{\underline{Q}}_{\Delta I} & \underline{\underline{Q}}_{\Delta\Delta} \end{pmatrix}, \underline{\underline{Q}}_{\Sigma\pi} \equiv \begin{pmatrix} \underline{\underline{Q}}_{I\pi} \\ \underline{\underline{Q}}_{\Delta\pi} \end{pmatrix}$$
$$\underline{\underline{Q}}_{\pi\Sigma} \equiv \begin{pmatrix} \underline{\underline{Q}}_{\pi I} & \underline{\underline{Q}}_{\pi\Delta} \end{pmatrix}, \underline{\underline{Q}}_{\pi\pi} \equiv \begin{pmatrix} \underline{\underline{Q}}_{\pi\pi} \end{pmatrix} \tag{9.9}$$

A. The Local DVS-software for $\underline{\underline{S}}$

Let $\underline{w}_\Delta \in W$, and recall Eq. (7.1); then:

$$\left(\underline{\underline{S}}\underline{w}\right)_\gamma = \underline{\underline{Q}}_{\Delta\Delta}\underline{w}_\Delta - \underline{\underline{Q}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{\sim1}\underline{\underline{a}}'\underline{\underline{Q}}_{\Pi\Delta}\underline{w}_\Delta \tag{9.10}$$

In this equation the meaning of the terms $\underline{\underline{Q}}_{\Delta\Delta}\underline{w}_\Delta$ and $\underline{\underline{Q}}_{\Pi\Delta}\underline{w}_\Delta$ are clear since both $\underline{\underline{Q}}_{\Delta\Delta}$ and $\underline{\underline{Q}}_{\Pi\Delta}$ are well-defined linear transformations of W^γ into itself. Something similar happens when the operator $\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{\sim1}$ is applied to $\underline{\underline{a}}'\underline{\underline{Q}}_{\Pi\Delta}\underline{w}_\Delta$, since this is also a global linear transformation. It must also be understood that, when it is applied, the local vector $\left(\underline{\underline{a}}'\underline{\underline{Q}}_{\Pi\Delta}\underline{w}_\Delta\right)_\gamma$ has already been stored at processor γ and at each one of the other processors. Due to the global character of the operator $\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{\sim1}$ special software was developed for it.

A.1. The local DVS-software for $\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{\sim1}$

The local software that was developed is based on the next formula:

"Let $\underline{w}_\Pi \in W^r(\Pi)$ and $\underline{v}_\Pi = \left(\underline{\underline{A}}_{\Pi\Pi}\right)^{\sim1}\underline{w}_\Pi$, then

$$\underline{\underline{a}}'\left(\underline{\underline{A}}^t_{\pi\pi} - \underline{\underline{A}}^t_{\pi I}\left(\underline{\underline{A}}^t_{II}\right)^{\sim1}\underline{\underline{A}}^t_{I\pi}\right)\underline{v}_\pi =$$
$$\underline{w}_\pi - \underline{\underline{A}}^t_{\pi I}\left(\underline{\underline{A}}^t_{II}\right)^{\sim1}\underline{w}_I,\ \textit{and}\ \underline{\underline{j}}\underline{v}_\pi = 0 \tag{9.11}$$

together with

$$\underline{v}_I = \left(\underline{\underline{A}}^t_{II}\right)^{-1}\left(\underline{w}_I - \underline{\underline{A}}^t_{I\pi}\underline{v}_\pi\right) \qquad (9.12)$$

To apply this formula iteratively, at each processor γ it was necessary to develop local software capable of carrying out the following operations:

$$\underline{\underline{a}}'\left(\underline{\underline{Q}}_{\pi\pi} - \underline{\underline{Q}}_{\pi I}\left(\underline{\underline{Q}}_{II}\right)^{-1}\underline{\underline{Q}}_{I\pi}\right)\underline{v}_\pi = \underline{w}_\pi - \underline{\underline{a}}'\underline{\underline{Q}}_{\pi I}\left(\underline{\underline{Q}}_{II}\right)^{-1}\underline{w}_I \qquad (9.13)$$

and, once convergence has been achieved, the following *autonomous* operation is carried out:

$$\underline{v}_I = \left(\underline{\underline{Q}}_{II}\right)^{-1}\left(\underline{w}_I - \underline{\underline{Q}}_{I\pi}\underline{v}_\pi\right) \qquad (9.14)$$

We see here that except for $\underline{\underline{a}}'$ all the linear transformations involved are *autonomous* and can be expressed by means of local matrices defined in each processor. In the DVS-software that is the subject of this paper, such matrices were not constructed but we recognize that in some problems such an option may be more competitive,

B. The Local DVS-Software for $\underline{\underline{S}}^{-1}$

The local software that was developed is based on the next formula: *"When $\underline{w}_\Delta \in W(\Delta)$, then*

$$\underline{\underline{S}}^{-1}\underline{w} = \left(\underline{\underline{A}}^{-1}\underline{w}_\Delta\right)_\Delta \qquad (9.15)$$

Therefore, if $\underline{v} \in W^r$ is defined by the condition $\underline{\underline{A}}\underline{v} = \underline{w}_\Delta$ and it is written in the form $\underline{v} = \underline{v}_I + \underline{v}_\Delta + \underline{v}_\pi$, then $\underline{\underline{S}}^{-1}\underline{w} = \underline{v}_\Delta$. A more explicit form of the condition $\underline{v} \in W^r$ is $\underline{\underline{j}}\underline{v}_\pi = 0$. This latter condition together with the equation $\underline{\underline{A}}\underline{v} = \underline{w}_\Delta$ gives rise to a global problem whose solution, in the parallel software we have developed, was based on the iterative scheme: *"Let $\underline{w}_\Delta \in W(\Delta)$ and $\underline{v}_\Delta \equiv \underline{\underline{S}}^{-1}\underline{w}_\Delta = \left(\underline{\underline{A}}^{-1}\underline{w}_\Delta\right)_\Delta$, then at processor γ*:

$$\underline{\underline{a}}'\left(\underline{\underline{Q}}_{\pi\pi} - \underline{\underline{Q}}_{\pi\Sigma}\left(\underline{\underline{Q}}_{\Sigma\Sigma}\right)^{-1}\underline{\underline{Q}}_{\Sigma\pi}\right)\underline{v}_\pi = -\underline{\underline{a}}'\underline{\underline{Q}}_{\pi\Sigma}\left(\underline{\underline{Q}}_{\Sigma\Sigma}\right)^{-1}\underline{w}\ , and\ \underline{\underline{j}}\underline{v}_\pi = 0 \qquad (9.16)$$

Once $\underline{v}_\pi$ has been obtained, $\underline{v}_\Delta$ is given by

$$\underline{\underline{S}}^{-1}\underline{w}_\Delta = \underline{v}_\Delta = \left(\left(\underline{\underline{Q}}_{\Sigma\Sigma}\right)^{-1}\left(\underline{w}_{\Sigma\Sigma} - \underline{\underline{Q}}_{\Sigma\pi}\underline{v}_\pi\right)\right)_\Delta \qquad (9.17)$$

At processor γ that is being considered, Eqs. (9.16) and (9.17) are:

$$\underline{\underline{a}}'\left(\underline{\underline{Q}}_{\pi\pi} - \underline{\underline{Q}}_{\pi\Sigma}\left(\underline{\underline{Q}}_{\Sigma\Sigma}\right)^{-1}\underline{\underline{Q}}_{\Sigma\pi}\right)\underline{v}_\pi = -\underline{\underline{a}}'\underline{\underline{Q}}_{\pi\Sigma}\left(\underline{\underline{Q}}_{\Sigma\Sigma}\right)^{-1}\underline{w}\ , and\ \underline{\underline{j}}\underline{v}_\pi = 0 \qquad (9.18)$$

and

$$\underline{v}_\Delta = \left(\left(\underline{\underline{Q}}_{\Sigma\Sigma}\right)^{-1}\left(\underline{w}_\Sigma - \underline{\underline{Q}}_{\Sigma\pi}\underline{v}_\pi\right)\right)_\Delta \qquad (9.19)$$

respectively.

C. Applications of the Conjugate Gradient Method (CGM)

There are three main instances in which CGM was applied: *i*) to invert $\underline{\underline{A}}_{\Pi\Pi}$; *ii*) to invert $\underline{\underline{A}}$; *iii*) to solve iteratively the *global equation* -such equation may be: either Eq. (6.5), Eq. (6.6), Eq. (6.8) or Eq. (6.10), depending on the *DVS-algorithm* that is applied-. Furthermore, it should be mentioned that the inverses of the *local-matrices*: $\underline{\underline{Q}}_{II}$ and $\underline{\underline{Q}}_{\Sigma\Sigma}$ can either be obtained by direct or by iterative methods; in the *DVS-software* here reported, this latter option was chosen and CGM was also applied at that level.

Numerical Results

In the numerical experiments that were carried out to test the *DVS-software*, the boundary-value problem for static elasticity introduced in the standar discretization was treated. In this paper only the DVS-BDDC algorithm has been tested. Work is underway to test the other *DVS-algorithms*, albeit similar results are expected for them. The elastic material was assumed to be homogeneous; so, the Lamé parameters were assumed to be constant and their values were taken to be

Table 1. Numerical Results

Number of Subdomains = Number of processors	DoF.	Nodes by Subdomain	Primal Nodes	Processing Time in seconds	Parallel efficiency $\left(\frac{p_{\min}}{p_{\max}}\bullet s\right)$x100	Speed up $s=\frac{T(p_{\min})}{T(p_{\max})}$	Norm of error $\left\|\underline{e}\right\|_{\circ}$
8	22,244,625	941,192	583	14,959	1	1	0.0263
27	21,904,152	274,625	2,312	5,882	75%	2.543	0.018
64	22,244,625	117,649	5,211	2,676	70%	5.59	0.029
125	21,904,152	59,319	9,184	1,212	79%	12.342	0.011
216	22,936,119	35,937	14,525	703	79%	21.280	0.010
343	22,244,625	21,952	20,628	406	86%	36.845	0.010
512	23,641,797	13,824	27,391	242	97%	61.814	0.011
729	23,287,176	10,648	36,800	183	90%	81.74	0.010
1000	23,641,797	8,000	46,899	136	88%	109.992	0.009
1331	22,936,119	5,832	57,100	96	94%	155.823	0.010
1728	20,903,613	4,096	66,671	89	78%	168.078	0.009
2197	21,904,152	3,375	80,352	64	85%	233.734	0.008
2744	22,244,625	2,744	94,471	51	86%	293.313	0.009

$$\lambda = \frac{Ev}{(1+v)(1-2v)} = 29.6412\text{x}10^9\,\frac{N}{m^2}$$
$$\mu = \frac{E}{2(1+v)} = 27.3611\text{x}10^9\,\frac{N}{m^2}$$

These values correspond to a class of cast iron (for further details about such a material see, http://en.wikipedia.org/wiki/Poisson's_ratio) whose *Young modulus*, E, and *Poison ratio*, v, are:

$$E = 68.95\text{x}10^9\,\frac{N}{m^2}\ \textit{and}\ v = 0.26$$

The domain $\Omega\subset R^3$ that the homogeneous-isotropic linearly-elastic solid considered occupies is a unitary cube. The boundary-value problem considered is a *Dirichlet problem*, with homogeneous boundary conditions, whose exact solution is:

$$\underline{u} = (\sin\pi x\sin\pi y\sin\pi z,\ \sin\pi x\sin\pi y\sin\pi z,\ \sin\pi x\sin\pi y\sin\pi z) \tag{10.1}$$

The *fine-mesh* that was introduced consisted of $(193)^3$ = 7,189,057 cubes, which yielded $(194)^3$ = 7,301,384 *original-nodes*.

The *coarse-mesh* consisted of a family of subdomains $\{\Omega_1, ..., \Omega_E\}$, whose interfaces constitute the *internal-boundary* Γ. The number E of subdomains was varied taking successively the values 8, 27, 64, 125, 216, 343, 512 and so on up to 2,744. The total number of *derived-nodes* and corresponding number of *degrees-of-freedom* are around 7.5 x 10^6 and 2.5 x 10^6, respectively. The constraints that were imposed consisted of continuity at *primal-nodes*; in every one of the numerical experiments all the nodes located at edges and vertices of the *coarse mesh* were taken as *primal-nodes*. In this manner, the total number of *primal-nodes* varied from a minimum of 583 to a maximum of 94,471. Thereby, it should be mentioned that these conditions granted that at each one of the numerical experiments the matrix $\underline{\underline{A}}$ was positive definite and possessed a well-defined inverse.

All the codes were developed in C++ and MPI was used. The computations were performed at the Mitzli Supercomputer of the National Autonomous University of Mexico (UNAM), operated by the DGTIC. All calculations were carried out in a 314-node cluster with 8 processors per node. The cluster consists 2.6 GHz Intel Xeon Sandy Bridge E5-2670 processors with 48 GB of RAM.

As it was exhibited in the analysis of the operations, the transmission of information between different processors exclusively occurs when the *average-operators* $\underline{\underline{a}}$ and $\underline{\underline{a}}'$ are applied. In a first version of the software reported in the present paper such exchange of information was carried out through a *master-processor*, which is time expensive. However, the efficiency of the software (as a parallelization tool) improved very much when the participation of the *master-processor* in the communication and exchange of information process was avoided. In its new version, the *master-processor* was eliminated altogether. A Table summarizing the numerical results follows.

It should be noticed that the *computational efficiency* is very high, reaching a maximum value of 96.6%. Furthermore, the efficiency increases as the number of processors increases, a commendable feature for software that intends to be top as a tool for programming the largest supercomputers available at present.

Conclusions

1. This paper contributes to further develop *non-overlapping discretization methods* and the *derived-vector approach (DVS)*, introduced by I. Herrera and co-workers (Herrera *et al*., 2014), (Herrera and Rosas-Medina, 2013), (Carrillo-Ledesma *et al*., 2013), (Herrera and Yates, 2011), (Herrera, 2007), (Herrera, 2008), (Herrera and Yates, 2010) and (Herrera and Yates, 2011);

2. A procedure for transforming *overlapping discretizations* into *non-overlapping* ones has been presented;

3. Such a method is applicable to symmetric and non-symmetric matrices;

4. To illustrate the procedures that are needed for constructing software based on non-*overlapping discretizations*, software suitable to treat problems of isotropic static elasticity has been developed; and

5. The software so obtained has been numerically tested and the high efficiency, as a parallelization tool, expected from DVS software has been experimentally confirmed.

The main general conclusion is that the *DVS* approach and *non-overlapping discretizations* are very adequate tools for applying highly parallelized hardware to treat the partial differential equations occurring in systems of science and engineering.

Acknowlegement

We thank DGTIC-UNAM for the extensive support we received to perform the computational experiments presented in Table 1.

References

Carrillo-Ledesma A., Herrera I., de la Cruz Luis Miguel, 2013, Parallel Algorithms for Computational Models of Geophysical Systems". *Geofísica Internacional*, 52, 3, pp., 293-309.

DDM Organization, Proceedings of 22 International Conferences on Domain Decomposition Methods www.ddm.org, 1988-2014.

Dohrmann C.R., 2003, A preconditioner for substructuring based on constrained energy minimization. *SIAM J. Sci. Comput*., 25, 1, 246-258.

Farhat Ch., Roux F., 1991, A method of finite element tearing and interconnecting and its parallel solution algorithm. *Internat. J. Numer. Methods Engrg*., 32:1205-1227.

Farhat C., Lessoinne M., Pierson K., 2000, A scalable dual-primal domain decomposition method, *Numer. Linear Algebra Appl.*, 7, pp 687-714.

Farhat C., Lessoinne M., LeTallec P., Pierson K., Rixen D., 2001, FETI-DP a dual-primal unified FETI method, Part I: A faster alternative to the two-level FETI method. *Int. J. Numer. Methods Engrg*., 50, pp 1523-1544.

Herrera I., New Formulation of Iterative Substructuring Methods without Lagrange Multipliers: Neumann-Neumann and FETI, NUMER METH PART D E 24(3) pp 845-878, 2008 (Published on line Sep 17, 2007) DOI 10.1002/num 20293.

Herrera I., Theory of Differential Equations in Discontinuous Piecewise-Defined-Functions, NUMER METH PART D E, 23(3), pp 597-639, 2007. DOI 10.1002/num 20182.

Herrera I., de la Cruz L.M., Rosas-Medina A., Non Overlapping Discretization Methods for Partial, Differential Equations. NUMER METH PART D E, 30: 1427-1454, 2014, DOI 10.1002/num 21852. (Open source)

Herrera I., Pinder G.F., 2012, Mathematical Modelling in Science and Engineering: An axiomatic approach, John Wiley, 243p.

Herrera I., Rosas-Medina A., 2013, The Derived-Vector Space Framework and Four General Purposes Massively Parallel DDM

Algorithms", EABE (Engineering Analysis with Boundary Elements), 37, pp-646-657.

Herrera I., Yates R.A., The Multipliers-Free Dual Primal Domain Decomposition Methods for Nonsymmetric Matrices" NUMER. METH. PART D. E. 27(5) pp. 1262-1289, 2011. (Published on line April 28, 2010) DOI 10.1002/num.20581.

Herrera I., Yates R.A., The Multipliers-free Domain Decomposition Methods, NUMER. METH. PART D. E. 26: 874-905 July 2010 (DOI 10.1002/num. 20462)

Herrera I., Yates R.A., The Multipliers-Free Dual Primal Domain Decomposition Methods for Nonsymmetric Matrices, NUMER. METH. PART D. E. 27(5) pp. 1262-1289, 2011. DOI 10.1002/Num. 20581.

Mandel J., Tezaur R., 1996, Convergence of a substructuring method with Lagrange multipliers. *Numer. Math,* 73, 4, 473-487, .

Mandel J., Dohrmann C.R., 2003, Convergence of a balancing domain decomposition by constraints and energy minimization, Numer. *Linear Algebra Appl*., 10, 7, 639-659.

Mandel J., Dohrmann C.R., Tezaur R., 2005, *An algebraic theory for primal and dual substructuring methods by constraints*, *Appl. Numer. Math.*, 54, 167-193.

President's Information Technology Advisoty Committee: PITAC, Computational Science: Ensuring America´s Competitiveness, Report to the President June 2005. 104 p. www.nitrd.gow/pitac

Toselli A., Widlund O., 2005, Domain decomposition methods- Algorithms and Theory, Springer Series in Computational Mathematics, Springer-Verlag, Berlin, 450p.