Improving the Numerical Accuracy of Parallel Programs by Data Mapping

Farah Benmouhoub[1], Matthieu Martel[1], and Pierre-Loic Garoche[2]

[1] LAMPS Laboratory, University of Perpignan, 52 Av Paul Alduy
Perpignan, France, 66860.
[2] DTIS, ONERA, 2 Av Edouard Belin,
Toulouse, France, 31000.
[1]{first.last}@univ-perp.fr,[2]pierre-loic garoche@onera.fr

Abstract. The first objective of parallelization is to speed up the program execution. Typically, a program is split into multiple parts that are computed on different computation cores. A usual approach is to balance the load of each core, splitting the computation evenly among them.
When the program performs computations in floating-point arithmetic, we should pay extra attention to their numerical accuracy. Indeed, the floating-point numbers are a finite approximation of real numbers, they are therefore prone to accuracy problems due to the accumulated round-off errors. Concerning the numerical accuracy, parallelism introduces additional problems due to the order of operations between several computation units.
Rather than focusing on balancing the load, we focus here on a proper split of the problem driven by the numerical accuracy of the computation. In this paper, we describe a new technique that relies on static analysis by abstract interpretation, and which aims at improving the numerical accuracy of computations by dividing the problem, between computation units, according to the order of magnitude of data.

Keywords: Numerical accuracy, Static analysis, Mapping, scientific computing.

1 Introduction

Scientific computing is typically performed with floating-point arithmetics and therefore sensitive to associated errors; and this problem tends to increase with parallelism. To cope with this issue, we aim at improving the accuracy of computation [3] using a new technique based on static analysis by abstract interpretation. In floating-point computations, in addition to rounding errors, the computations order may also affect the accuracy of the results. In this context, different summation algorithms are possible where the result depends on the order of data used to compute this summation as presented in the examples given in Figure 1.

To better illustrate, we can take an example of calculating the sum of three values x, y and z, where $x = 10^9$, $y = -10^9$ et $z = 10^{-9}$, we obtain:

$$((x + y) + z) = ((10^9 - 10^9) + 10^{-9}) = 10^{-9} \quad (1)$$

$$(x + (y + z)) = (10^9 + (-10^9 + 10^{-9})) = 0 \quad (2)$$

The equality as well as addition represent here the operator performed with floating point arithmetics. We note that, for the same values of x, y and z, and for the same arithmetic operation, we get two different results because of parsing the three values differently.

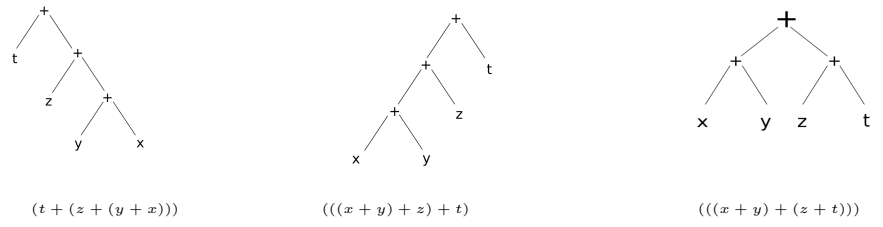

Fig. 1. The different sums possible.

The key idea of this work is to detect potential ordering of scalars that could lead to an optimization of the numerical accuracy of the computation. In the lack of accuracy of the above example, the issue was caused by computation involving scalars of different orders of magnitude. Roughly speaking, in order to keep the computation accurate, one need to know if the variables can be ordered with respect to their order of magnitude, starting summations with the smallest elements.

We propose to rely on static analysis to detect such arrangements of matrix coefficients: analyzing the data or the program, we would like to detect the order of magnitude of each scalar and the ordering (increasing, decreasing, balanced) of each part of data assigned to each computation unit. Once this ordering is accurately computed, one can choose an appropriate summation algorithm (left to right, right to left, balanced) and obtain more accurate floating point results.

In the case of parallel programs [10], we aim at specializing the code of each process depending on its data, instead on focusing only on load balancing between computation cores. This specialization is based on data mapping [7] [8]. The idea is to assign to each processor sets of data that can be summed toghether accurately. This corresponds to a certain parsing of the sum. To do so, we start with a static analysis of data and next we distribute data accordingly among the computation units. To illustrate this technique, we will apply it to an iterative method for solving a linear system of the form $Ax = b$. To keep the presentation simple we used the simplest iterative scheme: Jacobi's method.

In this article, we describe the different stages of our technique. First of all, given a sequence of values $s = x_1, x_2, \ldots x_n$, the property indicating how the elements of s are ordered is called the gradient of s. We denote $grad(s)$ this property. The gradient $grad(s)$ may be either increasing, decreasing, constant or unknown respectively denoted by $grad(s) \in \{\nearrow, \searrow, \rightarrow, \top\}$.

The first step of our method is to perform a static analysis of the matrix A and the vector x in order to be able to identify the sign and the gradient of the different blocks. After this identification, each block is assigned to a computation unit with a well-chosen summation algorithm.

This article is organized as follows. Section 2 presents briefly Jacobi's method, as well as existing works parallelizing it; we also justify our choice for this method. Section 3 presents our contribution: we detail our technique with its different steps. Section 4 details our motivational example. Section 5 describes the experimental results obtained during the measurement of the efficiency of our technique. Finally, in Section 6 we conclude and discuss about the future work in the continuation of the present article.

2 Jacobi's Method

2.1 A simple Iterative Algorithm to Solve Linear Systems

The Jacobi method is a well known numerical method used to solve linear systems of n equations with n unknowns. We choose it for its simplicity, and as a first algorithm on which to apply our methodology. In this method, an initial approximate solution x^0 is selected and through an iterative procedure the algorithm tries to find the real solution x.

For more description let us consider the following system of n linear equations $Ax = b$, where:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \textit{and} \quad x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

The computation of the solution at each iteration is given by Equation (3) below:

$$x_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1,\ i \neq j}^{n} a_{ij} x_j^k \right). \qquad i = 1, \ldots, n, \quad a_{ii} \neq 0. \tag{3}$$

Note that Jacobi's method is stable whenever the matrix A is strictly diagonally dominant (cf. Equation 4), ie. on each line, the absolute value of the diagonal term is greater than the sum of absolute values of the other terms:

$$\forall i \in 1, \ldots, n, \qquad |a_{ii}| > \sum_{i \neq j} |a_{ij}|. \tag{4}$$

Our choice for this method is related to the existence of the summation operator, i.e. a sum is done during the computation of each iterate. Since we manipulate floating-point numbers, this sum may become wrong because of accumulated errors, for an example, if we sum the small values with the large ones an absorption may arise and consequently a problem of accuracy. To limit this problems it is recommended to sum the values increasingly (in absolute value).

2.2 Parallelisation of Jacobi's Method

Several works have been focused on parallelizing Jacobi algorithms. Luke and Park [9] consider two parallel Jacobi algorithms for computing the singular value decomposition of an $n \times n$ matrix. By relating the algorithms to the cyclic-by-rows Jacobi method, they prove convergence of one of the algorithm for odd values of n and in the general case for the second algorithm.

Zhou and Brent [1] show the importance of sorting columns by norms in each sweep for one-sided Jacobi SVD computation. They describe two parallel Jacobi orderings. These orderings generate $n(n-1)/2$ different index pairs and sort column norms at the same time. The one-sided Jacobi SVD algorithm using these parallel orderings converges in about the same number of sweeps as the sequential cyclic Jacobi algorithm.

Coope and Macklem [5] present how to efficiently use parallel and distributed computing platforms when solving derivative-free optimization problems with the Jacobi algorithm. Convergence is achieved by introducing an elementary trust region subproblem at synchronization steps in the algorithm.

All these works focused on the parallelisation of Jacobi's method. Some are focused on the improvement of the convergence of the algorithm, eg. when computing a singular value decomposition. Some others try to make the best use of the parallel architecture performance. However, none of them addressed the issue of the numerical accuracy of computations.

Our past results [6] show that improving the accuracy of computation also led to acceleration of convergence for iterative algorithms. Our motivation is therefore to parallelize the Jacobi's method focusing first on accuracy and obtaining, as a side result, a better convergence.

3 Accuracy bassed Mapping

3.1 Greedy Algorithm

Our contribution relies on static analysis for the detection of lines, columns or blocks of the matrix that could be efficiently optimized with respect to numerical accuracy. More precisely, we identify a group of data $x = x_1, x_2, \dots x_n$ accordingly to theirs signs and magnitudes, we call this property of x the gradiant of x.

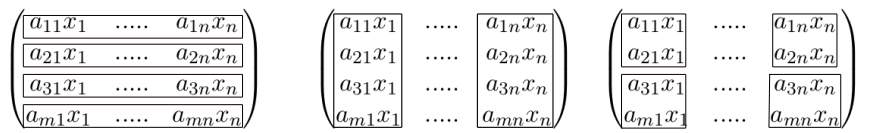

Fig. 2. Different parallelisation of Jacobi's method.

As mentioned in Section 1, we consider four different gradiants: $\nearrow$ (increasing), $\searrow$ (decreasing), $\rightarrow$ (balanced) or $\top$ (unknown), given a large sequence $x = x_1, x_2, \ldots x_n$ corresponding to a line or column or block of the matrix (see Figure 2).

We oftenly cannot assign a single gradiant to the whole sequence x. In this case, we aim at assigning a sequence of gradiants to x. For example, let $x = 1, 2, 3, 4, 4, 3, 2, 1$ we aim at assigning the sequence $\{\nearrow, \rightarrow, \searrow\}$ of gradiants to x. To do so we use the greedy algorithm displayed in Algorithm 1.

Algorithm 1 Greedy algorithm

```
while (iteration < Nb_iterations) do
  while (position < (iteration × n) − 1) do
    Grad_position = position;
    Grad_x = compare(x[position], x[position + 1]);
    position = position + 1;
    New_Grad_x = compare(x[position], x[position + 1]);
    while ((Grad_x = New_Grad_x)  and  (position < (iteration × n) − 1))) do
      position = position + 1;
      Grad_x = New_Grad_x;
      New_Grad_x = compare(x[position], x[position + 1]);
    end while
  end while
  iteration=iteration+1;
  position=position+1;
end while
```

In our study to detect the blocks of the same sign and magnitude we use what we call the greedy algorithm. The idea is to compare the values of the vector scalars pair by pair, these values can be in decreasing, increasing or balanced order. The number of blocks is determined by the number of gradiants calculated by this algorithm. To do so, the algorithm takes as input the solution vector $x = x_1, x_2.....x_n$ at each iteration. The outputs of the algorithm are the gradiant of each block noted by $Grad\_x$ and the index of the component by which this block begins noted by $Grad\_position$. If we consider the example presented previously $x = 1, 2, 3, 4, 4, 3, 2, 1$ we obtain as an output the following two pieces of information for each block, $[Grad\_position = 1 : Grad\_x = \nearrow]$, $[Grad\_position = 4 : Grad\_x = \rightarrow]$, $[Grad\_position = 5 : Grad\_x = \searrow]$.

3.2 Static Analysis

In this section, we introduce our principal contribution, detecting increasing, decreasing or balance patterns in vectors in order to select to proper summation arrangment.

We introduce abstract domains able to detect such patterns, as properties over vector sets. Let $\mathbb{R}^n$ be a vector of size n and $\wp(\mathbb{R}^n)$ a set of such n-sized

vectors. This set is fitted with a partial order, the set inclusion. It is also a complete lattice.

Thanks to the framework of Abstract Intepretation [4] we can define different abstractions of this lattice, and combine them to characterize properties of sets of n-vectors.

Figure 3 sketches our hierachy of abstractions. A first abstraction *Sign* relies on the classical sign domains to detect whether all elements of the vector have the same sign. A second one, *Grad*(ient) is used to represent the increasing, decreasing or balanced nature of the vector scalars. This property is computed over a first abstraction representing values by their floating point exponent, ie. their order of magnitude. We now define precisely each abstraction step.

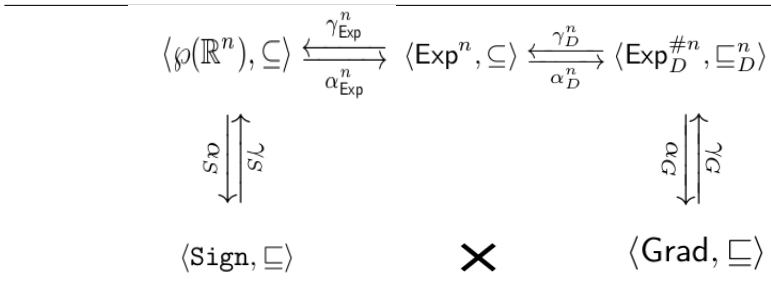

Fig. 3. Global diagram of abstractions.

3.3 Exp^n: order of magnitude of matrix elements

A first abstract domain represents a set of n-vectors $\wp(\mathbb{R}^n)$ by a vector of set of exponent $\mathsf{Exp}^n = (\wp(\mathbb{Z}))^n$. Exponents are defined as signed integers corresponding to the exponent of the real number they represent, ie. the power of 10 they have when expressed in scientific notation.

Let us first formalize this abstraction: the lattice $\langle \wp(\mathbb{R}), \subseteq, \cup, \cap, \emptyset, \mathbb{R} \rangle$ is abstracted by the lattice $\langle \wp(\mathbb{Z}), \subseteq, \cup, \cap, \emptyset, \mathbb{Z} \rangle$. Let $(\alpha_{\mathsf{Exp}}, \gamma_{\mathsf{Exp}})$ be the pair of abstraction and concretization functions defined as:

$$\begin{cases} \alpha_{\mathsf{Exp}} : \wp(\mathbb{R}) \to \wp(\mathbb{Z}) \\ \qquad\qquad X \mapsto \{\log_{10}(x) : x \in X)\} \\ \gamma_{\mathsf{Exp}} : \wp(\mathbb{Z}) \to \wp(\mathbb{R}) \\ \qquad\qquad Y \mapsto \{x \in \mathbb{R} | \log_{10}(x) \in Y)\} \end{cases} \tag{5}$$

Theorem 1 (Exp galois connection). *The pair of $(\alpha_{\mathsf{Exp}}, \gamma_{\mathsf{Exp}})$ is a Galois connection.*

$$\langle \wp(\mathbb{R}), \subseteq \rangle \underset{\alpha_{\mathsf{Exp}}}{\overset{\gamma_{\mathsf{Exp}}}{\leftrightarrows}} \langle \mathit{Exp}, \subseteq \rangle$$

Proof. Both domains are sets and the functions are defined element-wise: they are monotonic. $\alpha_{\mathsf{Exp}} \circ \gamma_{\mathsf{Exp}}(Y) = Y$ is reductive while $\gamma_{\mathsf{Exp}} \circ \alpha_{\mathsf{Exp}}(X) = \{x \in \mathbb{R} | \exists x' \in X, log_{10}(x) = log_{10}(x')\} \supseteq X$ is extensive. □

The abstraction function represents a set of value by the magnitude obtained with a log_{10} function. The concretization function is the associated operation to obtain a Galois connection. As an example, $\alpha_{\mathsf{Exp}}(\{1.10^4, 2.10^4, 3.10^4\}) = \{4\}$ since all these values share the same exponent 4.

We can now define the lift of this abstraction to sets of n-vectors in $\wp(\mathbb{R}^n)$. The lattice $\langle \wp(\mathbb{R}^n), \subseteq, \cup, \cap, \emptyset, \mathbb{R} \rangle$ is abstracted by the lattice $\mathsf{Exp}^n = \langle (\wp\mathbb{Z})^n, \subseteq^n, \cup^n, \cap^n, \emptyset^n, \mathbb{Z}^n \rangle$ where $\subseteq^n, \cup^n$, and $\cap^n$ denote the lift of classical set operators to n-vectors. Eg. $\forall x, y \in (\wp\mathbb{Z})^n, x \subseteq^n y$ iff $\forall i \in [1, n], x_i \subseteq y_i$. Similarly $\forall x, y \in (\wp\mathbb{Z})^n, \exists z \in (\wp\mathbb{Z})^n$, st. $z = x \cup^n y$ and $\forall i \in [1, n], z_i = x_i \cup y_i$.

Let us introduce the pair of function $(\alpha^n_{\mathsf{Exp}}, \gamma^n_{\mathsf{Exp}})$:

$$\begin{cases} \alpha^n_{\mathsf{Exp}} : \wp(\mathbb{R}^n) \to (\wp\mathbb{Z})^n \\ \qquad\qquad X \mapsto z \in (\wp\mathbb{Z})^n \text{ s.t. } \forall i \in [1, n], z_i = \alpha_{\mathsf{Exp}}(\{x_i | x \in X\}) \\ \gamma^n_{\mathsf{Exp}} : (\wp\mathbb{Z})^n \to \wp(\mathbb{R}^n) \\ \qquad\qquad z \mapsto \{x \in \mathbb{R}^n | \forall i \in [1, n], x_i \in \gamma_{\mathsf{Exp}}(z_i)\} \end{cases} \tag{6}$$

Theorem 2 (Exp^n galois connection). *The pair of $(\alpha^n_{\mathsf{Exp}}, \gamma^n_{\mathsf{Exp}})$ is a Galois connection.*

$$\langle \wp(\mathbb{R}^n), \subseteq \rangle \underset{\alpha^n_{\mathsf{Exp}}}{\overset{\gamma^n_{\mathsf{Exp}}}{\leftrightarrows}} \langle \mathit{Exp}^n, \subseteq^n \rangle$$

Proof. The two domains $\wp(\mathbb{R}^n)$ and $\wp(\mathbb{Z})^n$ are sets and the functions are defined element-wise for each vector: they are monotonic. $\alpha^n_{\mathsf{Exp}} \circ \gamma^n_{\mathsf{Exp}}(z) = z$ is reductive while $\gamma^n_{\mathsf{Exp}} \circ \alpha^n_{\mathsf{Exp}}(X) = \{x \in \mathbb{R}^n | \exists x' \in X \quad \forall i \in [1, n], \alpha_{\mathsf{Exp}}(x'_i) = \alpha_{\mathsf{Exp}}(x_i)\} \supseteq X$ is extensive. □

The example below represents the abstraction of a set of vectors $\wp(\mathbb{R}^n)$ by a vector of set of exponent $\wp(\mathbb{Z})^n$ using the abstract function α^n_{Exp}.

$$\left\{ \begin{pmatrix} 1000 \\ 100 \\ 10 \end{pmatrix}, \begin{pmatrix} 0.001 \\ 1 \\ 0.1 \end{pmatrix} \right\} \overset{\alpha^n_{\mathsf{Exp}}}{\longrightarrow} \left\{ \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}, \begin{pmatrix} -3 \\ 0 \\ -1 \end{pmatrix} \right\}$$

3.4 $\mathsf{Exp}^{\#n}$: Abstraction of Exponants in Scalar Words

We need to further abstract the vectors of set of exponents $\mathsf{Exp}^n = (\wp\mathbb{Z})^n$ we built previous section. Since each index of these vectors is a set of integers, one can rely on the state of the art of abstract domains to represent these sets.

Let $\langle D, \sqsubseteq_D \rangle$ be a sound abstraction of $\langle \wp(\mathbb{Z}, \subseteq \rangle$ associated with a proper Galois connection (α_D, γ_D). We define the set $\mathsf{Exp}_D^{\#n} = D^n$ as the n-vector of D elements. Each operator is the lift of domain D operators to vectors. Eg. $\forall x, y \in \mathsf{Exp}_D^{\#n}, x \sqsubseteq_D^n y$ iff $\forall i \in [1, n], x_i \sqsubseteq_D y_i$.

We introduce the following abstraction:

$$\langle \mathsf{Exp}^n, \subseteq \rangle \underset{\alpha_D^n}{\overset{\gamma_D^n}{\leftrightarrows}} \langle \mathsf{Exp}_D^{\#n}, \sqsubseteq_D^n \rangle \tag{7}$$

where $\alpha_D^n(x) = z$ with $\forall i \in [1, n], z_i = \alpha_D(x)$. Similarly, $\gamma_D^n(z) = x \in \mathsf{Exp}^n$ with $\forall i \in [1, n], x_i \subseteq \gamma_D(z_i)$.

We propose to instantiate this abstraction by two basic abstract domains:

1. Kildall's constants domain [] $D = \mathbb{Z} \cup \{\bot, \top\}$
2. Intervals $D = ((\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{+\infty\})) \cup \{\bot\}$

3.5 Abstraction in Gradient

To abstract the gradient from the data, we define a new comparison relation according to the order of magnitude of the data noted by $\leq^{\#}$, such that we associate to a set of values $(x_1 \cdot x_2 \cdot .... \cdot x_n)$ one of the following five values $\{\bot, \nearrow, \searrow, \rightarrow, \top\}$.

$$\begin{aligned} Exp^{\#n} &= \text{Kildall}(\mathbb{Z})^n \qquad x_1 \leq^{\#} x_2 \\ &= \text{Interv}(\mathbb{Z})^n \quad [a, b] \leq^{\#} [c, d] \\ &\qquad\qquad\qquad\quad \leq_1^{\#}: b \leq c \\ &\qquad\qquad\qquad\quad \leq_2^{\#}: a \leq d \end{aligned}$$

$$\alpha_G(x_1 \cdot x_2 \cdot .... \cdot x_n) = \begin{cases} \nearrow & \text{if} \quad x_1 \leq^{\#} x_2 \cdot ....\cdot \leq^{\#} x_n, \\ \searrow & \text{if} \quad x_1 \geq^{\#} x_2 \cdot ....\cdot \geq^{\#} x_n, \\ \rightarrow & \text{if} \quad x_1 =^{\#} x_2 \cdot ...\cdot =^{\#} x_n, \\ \top & \text{Otherwhise.} \end{cases}$$

Where α_G is the abstract function which takes a sequence of real numbers $(x_1 \cdot x_2 \cdot .... \cdot x_n)$ and computes its gradient, ie. if the vector scalars is ordered in increasing order defined in this section by $\leq^{\#}$, the gradient is represented by $\nearrow$. Conversely, if the vecteur scalars is in the decreasing order the gradient takes the value $\searrow$. Similarly for the balanced vector scalars where the gradient takes the value $\rightarrow$, and when the values of the vector scalars are incomparable the gradient will be equal to $\top$.

4 Case Studies

4.1 Flexion of a Beam

The first example consists of a physical problem arising in Mechanics which concerns the flexion of an $1D$ elastic beam with Dirichlet boundary conditions on

its extremities [2]. The discretization of this kind of problem is based on the finite element method (FEM) that was usually used to solve complicated problems in engineering. The representative diagram of our case study is presented in Figure 4.

where u is the displacement such as $u_1 = \alpha$ and $u_{N+1} = \beta$, α and β the ex-

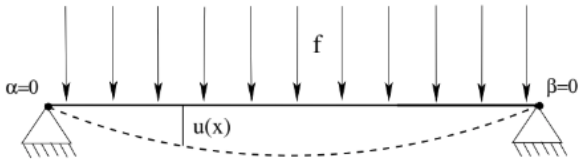

Fig. 4. Representation of the flexion of an 1D beam.

tremities where the beam is fixed such as ($\alpha = \beta = 0$). f is a constant vertical force acting on the domain interval $\Omega = [0, 1]$. A formalization of our problem can be given as follows:

$$\begin{cases} u''(x) = f \quad \forall x \in]0, 1] \\ u(0) = \alpha \quad \text{and} \quad u(1) = \beta \end{cases}$$

In order to obtain a linear system to resolve, we need to do the discetisation of the mesh. So, first we have to introduce the mesh of the domain $\Omega = [0, 1]$ by considering $N + 1$ nodes $\{x_i, i = 1, .., N + 1\}$ of the interval $[0, 1]$ with $x_1 = 0$, $x_{N+1} = 1$ and $x_{i+1} = x_i + h_i$, for $i = 1, .., N$. Next, the domain $[0, 1]$ is discretized into N intervals (x_i, x_{i+1}) that are the finite elements of size h_i. Finally, after calculations and substitution of the known values (u_1, u_{N+1}) we obtain the following tridiagonal system.

For our experiments, the values of h are computed automatically with the

$$\begin{pmatrix} \frac{1}{h_1} + C & \frac{-1}{h_1} & & & \\ \frac{-1}{h_1} & \frac{1}{h_1} + \frac{1}{h_2} & \frac{-1}{h_2} & & \\ & \ddots & \ddots & \ddots & \\ & \frac{-1}{h_{N-1}} & \frac{1}{h_{N-1}} + \frac{1}{h_N} & \frac{-1}{h_N} & \\ & & \frac{-1}{h_N} & \frac{1}{h_N} + C & \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \\ u_{N+1} \end{pmatrix} = \frac{f}{2} \begin{pmatrix} h_1 + C\alpha \\ h_1 + h_2 \\ \vdots \\ h_{N-1} + h_N \\ h_N + C\beta \end{pmatrix}$$

specificity of obtaining symmetrical values of h. More precisely, we initilize the first and the last values of h and we compute the others so that $h[n-i-1] = h[i]$. We suppose that the value of the penalization C is equal to 10^6 and the vertical

force $f = -20N/m^2$.
We remind that our subject is to create linear systems of different size N which modelize the flexion of a $1D$ beam. Then, we calculated the real solution X using the Jacobi's method.
Once the solution is calculated, we abstract the values of the solution vector to exponent as presented in Section 3.3. After abstraction we applied the greedy algorithm in order to found the gradient of the values, more precisely order of magnitude of this values.

4.2 Example

To better explain, we take an example of a matrix of N=4. First, we generate the system corresponding of the flexion of $1D$ beam. Then, We compute the solution of this linear system by the Jacobi's method, and we associate to each value of the solution vector its floating point exponent using the abstract function $\dot{\alpha_{Exp}}$ presented in the Section 3.2.

$$A = \begin{pmatrix} 11778.279410 & -1778.279410 \\ -1778.279410 & 3556.558820 \end{pmatrix}, \quad b = \begin{pmatrix} 0.000662 \\ 0.001125 \end{pmatrix}$$

$$x^1 = \begin{pmatrix} \text{-5.623062e-07} \Rightarrow Exp^1 = \text{-7} \\ \text{-3.162326e-06} \Rightarrow Exp^2 = \text{-6} \end{pmatrix} \quad x^2 = \begin{pmatrix} \text{-1.039753e-06} \Rightarrow Exp^1 = \text{-6} \\ \text{-3.443479e-06} \Rightarrow Exp^2 = \text{-6} \end{pmatrix}$$

$$x^3 = \begin{pmatrix} \text{-1.082201e-06} \Rightarrow Exp^1 = \text{-6} \\ \text{-3.682203e-06} \Rightarrow Exp^2 = \text{-6} \end{pmatrix} \quad x^4 = \begin{pmatrix} \text{-1.118244e-06} \Rightarrow Exp^1 = \text{-6} \\ \text{-3.703427e-06} \Rightarrow Exp^2 = \text{-6} \end{pmatrix}$$

$$x^5 = \begin{pmatrix} \text{-1.121448e-06} \Rightarrow Exp^1 = \text{-6} \\ \text{-3.721448e-06} \Rightarrow Exp^2 = \text{-6} \end{pmatrix} \quad x^6 = \begin{pmatrix} \text{-1.124169e-06} \Rightarrow Exp^1 = \text{-6} \\ \text{-3.723050e-06} \Rightarrow Exp^2 = \text{-6} \end{pmatrix}$$

$$x^7 = \begin{pmatrix} \text{-1.124169e-06} \Rightarrow Exp^1 = \text{-6} \\ \text{-3.723050e-06} \Rightarrow Exp^2 = \text{-6} \end{pmatrix}$$

Once this step is over, we apply a second abstraction noted by $Grad$(ient) on the exponents generated by the first abstraction in order to represent the increasing, decreasing or balanced nature of the vector scalars.

$$x^1 = \begin{pmatrix} Exp^1 = \text{-7} \\ Exp^2 = \text{-6} \end{pmatrix} \Rightarrow grad = \nearrow \qquad x^2 = \begin{pmatrix} Exp^1 = \text{-6} \\ Exp^2 = \text{-6} \end{pmatrix} \Rightarrow grad = \rightarrow$$

$$x^3 = \begin{pmatrix} Exp^1 = \text{-6} \\ Exp^2 = \text{-6} \end{pmatrix} \Rightarrow grad = \rightarrow \qquad x^4 = \begin{pmatrix} Exp^1 = \text{-6} \\ Exp^2 = \text{-6} \end{pmatrix} \Rightarrow grad = \rightarrow$$

$$x^5 = \begin{pmatrix} Exp^1 = \text{-6} \\ Exp^2 = \text{-6} \end{pmatrix} \Rightarrow grad = \rightarrow \qquad x^6 = \begin{pmatrix} Exp^1 = \text{-6} \\ Exp^2 = \text{-6} \end{pmatrix} \Rightarrow grad = \rightarrow$$

$$x^7 = \begin{pmatrix} Exp^1 = \text{-6} \\ Exp^2 = \text{-6} \end{pmatrix} \Rightarrow grad = \rightarrow$$

5 Experimental Results

In this section, we present the experimental results of our study concerning the numerical accuracy of computations using a new technique discussed in this article. First, we want to improve the effeciency of our technique in the detection of scalars that have the same sign and gradiant which can be grouped in block and that could eventually lead to an optimization of the numerical accuracy of computations. The most critical case is to have equality between the number of blocks and the number of scalars. In other words, the ratio between the total number of blocks and the size of matrix is equal to 1. To be done, we generate different linear systems of the form $Ax = b$ that model the example previously described in Section 4, after that we solve this systems using Jacobi's method, then we applying our technique on the solution vector x. For measuring effectiveness of our technique we compute the average of gradiant of a set of matrix to compare the variation of these average with those of matrix sizes, Figure 5 represents the gradiants average corresponding to each matrix size from 10 to 1000.

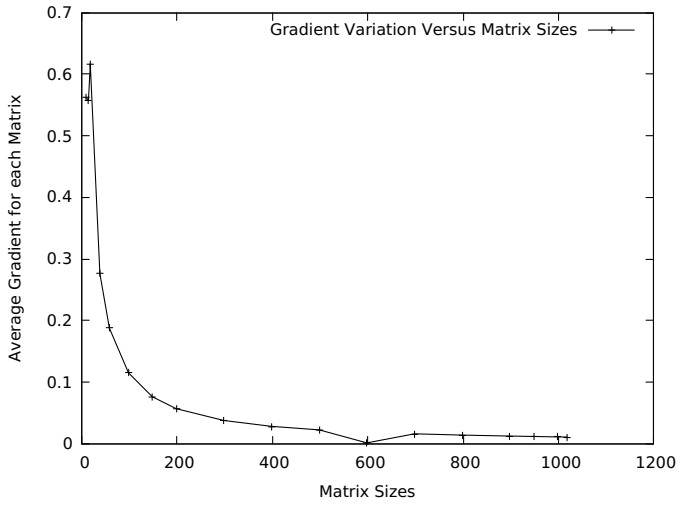

Fig. 5. Gradiants average for different matrix size from 10 to 1000.

We notice that for small matrix for example ($N = 100$) the average is around of 0.6, so we conclude that the number of blocks represents 60% of the size of matrix. We also notice that more the size of the matrix increases, more the average decreases. From these two remarks, we deduce that the efficiency of our technique is reached whene we handle large matrix.

Secondly, we want to know if for a given matrix we can generalize the division of scalars into lines, columns or blocks. As mentinnoed in Section 3, for a

linear system of the form $Ax = b$ we apply our technique to a solution vector $x = x_0, x_1, ...x_{n-1}$ at each iteration in order to group the data accordingly to theirs signs and magnitudes. We associate for each sequence of data a corresponding sequence of gradiants, and each block is represented by two informations: the index i of the component $x_i, \forall i \in [0, n-1]$ by which it starts and the gradiant associated to its sequence of scalars. For the sake of simplicity we consider a matrix of 16×16 with coefficients taken from a realistic example corresponding to the flexion of a beam developed in Section 4 the results of our study is given below:

$iteration1 : [0 :\rightarrow][2 :\searrow][3 :\rightarrow][4 :\searrow][5 :\rightarrow][9 :\nearrow][10 :\rightarrow][11 :\nearrow][12 :\rightarrow]$
$iteration2 : [0 :\rightarrow][1 :\searrow][2 :\nearrow][3 :\searrow][5 :\rightarrow][6 :\nearrow][7 :\searrow][8 :\rightarrow][10 :\nearrow][11 :\rightarrow][12 :\nearrow]$
$iteration3 : [0 :\rightarrow][1 :\searrow][2 :\nearrow][3 :\rightarrow][4 :\searrow][5 :\nearrow][6 :\searrow][7 :\nearrow][8 :\searrow][10 :\nearrow][11 :\rightarrow]$
$iteration4 : [0 :\searrow][1 :\nearrow][2 :\searrow][4 :\nearrow][5 :\searrow][7 :\nearrow][9 :\searrow][12 :\rightarrow]$
$iteration5 : [0 :\rightarrow][1 :\nearrow][2 :\searrow][3 :\nearrow][4 :\searrow][6 :\nearrow][7 :\searrow][8 :\nearrow][10 :\searrow]$
$iteration6 : [0 :\rightarrow][1 :\searrow][2 :\nearrow][3 :\searrow][4 :\rightarrow][6 :\nearrow][7 :\searrow][9 :\nearrow][12 :\searrow]$
$iteration7 : [0 :\rightarrow][1 :\searrow][2 :\rightarrow][3 :\searrow][4 :\nearrow][6 :\searrow][7 :\nearrow][8 :\searrow][10 :\nearrow][11 :\searrow][12 :\nearrow]$
$iteration8 : [0 :\rightarrow][1 :\nearrow][2 :\searrow][4 :\nearrow][5 :\searrow][7 :\nearrow][9 :\searrow]$
$iteration9 : [0 :\rightarrow][2 :\searrow][3 :\nearrow][4 :\searrow][6 :\nearrow][7 :\searrow][8 :\nearrow][10 :\searrow]$
$iteration10 : [0 :\rightarrow][1 :\searrow][2 :\nearrow][3 :\searrow][4 :\rightarrow][5 :\nearrow][7 :\searrow][9 :\rightarrow][10 :\nearrow][12 :\searrow]$
$iteration11 : [0 :\rightarrow][1 :\searrow][2 :\nearrow][3 :\searrow][4 :\nearrow][6 :\searrow][7 :\nearrow][8 :\searrow][10 :\nearrow][11 :\searrow][12 :\nearrow]$
$iteration12 : [0 :\rightarrow][1 :\nearrow][2 :\searrow][4 :\nearrow][5 :\searrow][6 :\rightarrow][8 :\nearrow][9 :\searrow]$
$iteration13 : [0 :\searrow][1 :\nearrow][2 :\searrow][3 :\nearrow][4 :\searrow][6 :\nearrow][7 :\searrow][8 :\nearrow][10 :\searrow]$
$iteration14 : [0 :\nearrow][1 :\searrow][2 :\nearrow][3 :\searrow][5 :\nearrow][7 :\searrow][9 :\nearrow][12 :\searrow]$
$iteration15 : [0 :\rightarrow][1 :\searrow][2 :\nearrow][3 :\searrow][4 :\nearrow][6 :\searrow][7 :\nearrow][8 :\searrow][10 :\nearrow][11 :\searrow][12 :\nearrow]$
$iteration16 : [0 :\rightarrow][1 :\nearrow][2 :\searrow][4 :\nearrow][5 :\searrow][7 :\nearrow][9 :\searrow]$
$iteration17 : [0 :\rightarrow][1 :\nearrow][2 :\searrow][3 :\nearrow][4 :\searrow][6 :\nearrow][7 :\searrow][8 :\nearrow][10 :\searrow]$
$iteration18 : [0 :\rightarrow][1 :\searrow][2 :\nearrow][3 :\searrow][5 :\nearrow][7 :\searrow][9 :\nearrow][12 :\searrow]$
$iteration19 : [0 :\nearrow][1 :\searrow][2 :\nearrow][3 :\searrow][4 :\nearrow][6 :\searrow][7 :\nearrow][8 :\searrow][10 :\nearrow][11 :\searrow][12 :\nearrow]$
$iteration20 : [0 :\searrow][1 :\nearrow][2 :\searrow][4 :\nearrow][5 :\searrow][7 :\nearrow][9 :\searrow]$
$iteration21 : [0 :\rightarrow][1 :\nearrow][2 :\searrow][3 :\nearrow][4 :\searrow][5 :\rightarrow][6 :\nearrow][7 :\searrow][8 :\rightarrow][9 :\nearrow][10 :\searrow]$
$iteration22 : [0 :\rightarrow][1 :\searrow][2 :\nearrow][3 :\searrow][5 :\nearrow][7 :\searrow][9 :\nearrow][12 :\searrow]$
$iteration23 : [0 :\rightarrow][1 :\searrow][2 :\nearrow][3 :\searrow][4 :\nearrow][6 :\searrow][7 :\nearrow][8 :\searrow][10 :\nearrow][11 :\searrow][12 :\nearrow]$
$iteration24 : [0 :\rightarrow][1 :\nearrow][2 :\searrow][4 :\nearrow][5 :\searrow][7 :\nearrow][9 :\searrow]$
$iteration25 : [0 :\rightarrow][2 :\searrow][3 :\nearrow][4 :\searrow][6 :\nearrow][7 :\searrow][8 :\nearrow][10 :\searrow]$

If we take iteration 11 as an example, we notice that after each 4 iterations we find the same sequence of blocks, ie. the same sequence of gradiants is associate to the sequence of scalars as it shown by iterations 15, 19, 23. In the same way, the iterations 16 and 14 are repeated after 4 iterations respectively given by the iterations 20, 24 and 18, 22.

6 Conclusion

References

1. B B. Zhou and Richard Brent. On parallel implementation of the one-sided jacobi algorithm forsingular value decompositions. pages 401–408, 02 1995.

2. Mikaël Barboteu, Nacera Djehaf, and Matthieu Martel. Numerically accurate code synthesis for gauss pivoting method to solve linear systems coming from mechanics. *Computers & Mathematics with Applications*, 77(11):2883–2893, 2019.
3. Farah Benmouhoub, Nasrine Damouche, and Matthieu Martel. Improving the numerical accuracy of high performance computing programs by process specialization. In Matthieu Martel, Nasrine Damouche, and Julien Alexandre Dit Sandretto, editors, *TNC'18. Trusted Numerical Computations*, volume 8 of *Kalpa Publications in Computing*, pages 11–23. EasyChair, 2018.
4. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Principles of Programming Languages*, pages 238–252, 1977.
5. I D. Coope and Mason Macklem. Parallel jacobi methods for derivative-free optimization on parallel or distributed processors. *The ANZIAM Journal*, 2004, 07 2005.
6. N. Damouche, M. Martel, and A. Chapoutot. Impact of accuracy optimization on the convergence of numerical iterative methods. In M. Falaschi, editor, *LOPSTR 2015*, volume 9527 of *Lecture Notes in Computer Science*, pages 143–160. Springer, 2015.
7. Yiannis Georgiou, Emmanuel Jeannot, Guillaume Mercier, and Adèle Villiermet. Topology-aware job mapping. *IJHPCA*, 32(1):14–27, 2018.
8. Torsten Hoefler, Emmanuel Jeannot, and Guillaume Mercier. An Overview of Process Mapping Techniques and Algorithms in High-Performance Computing. In Emmanuel Jeannot and Julius Zilinskas, editors, *High Performance Computing on Complex Environments*, pages 75–94. Wiley, June 2014.
9. F. T. Luk and H. Park. A proof of convergence for two parallel jacobi svd algorithms. *IEEE Trans. Comput.*, 38(6):806–811, June 1989.
10. D. Unat, A. Dubey, T. Hoefler, J. Shalf, M. Abraham, M. Bianco, B. L. Chamberlain, R. Cledat, H. C. Edwards, H. Finkel, K. Fuerlinger, F. Hannig, E. Jeannot, A. Kamil, J. Keasler, P. H. J. Kelly, V. Leung, H. Ltaief, N. Maruyama, C. J. Newburn, and M. Perics. Trends in data locality abstractions for hpc systems. *IEEE Transactions on Parallel and Distributed Systems*, 28(10):3007–3020, Oct 2017.