High Performance Computing and Neville elimination

Courses on Totally Positive and Totally Negative Matrices

Pedro Alonso Velázquez
Departamento de Matemáticas
Universidad de Oviedo

ALAMA, CIEM, marzo 2015

Contents

1. Introduction
2. Neville elimination
3. High Performance Computing
4. References

1 Introduction

- Neville elimination (NE) is an alternative procedure to Gaussian elimination (GE) for transforming a square matrix A into an upper triangular matrix U, preferable for some classes of matrices (totally positive matrices, sign-regular matrices, etc.) and when using pivoting strategies in parallel implementations.

- The totally positive matrices (TP) and their important subclass of almost strictly totally positive matrices (ASTP), play an important role in many applications of Approximation Theory, Statistics, Computer Aided Geometric Design (CAGD), etc.

- For these matrices and other related classes of matrices, NE has great advantages over GE in many important tasks, such as: solving linear systems, calculating singular values or eigenvalues, deriving tests to recognize total positivity, and others.

- In particular, we have analyzed different aspects about NE:
  - Error analysis (backward and forward),
  - High Performance Computing (HPC),
  - Iterative refinement,
  - Growth factor (numerical stability of a numerical algorithm),
  - Neville versus Gauss,
  - High Accuracy Relative (HRA),
  - Characterization of Almost Strictly Sign Regular matrices,
  - etc.

Contents

1. Introduction
2. Neville elimination
   - 2.1 Description
   - 2.2 Pivoting strategies
   - 2.3 Backward error analysis
3. High Performance Computing
4. References

2.1 Description

- If A is an $n \times n$ matrix, NE consists of at most $n-1$ successive major steps, resulting in a sequence of matrices as follows:

$$A = A^{(1)} \to \widetilde{A}^{(1)} \to A^{(2)} \to \widetilde{A}^{(2)} \to \cdots \to A^{(n)} = \widetilde{A}^{(n)} = U, \qquad (1)$$

  where U is an upper triangular matrix.

- For each t, $1 \leq t \leq n$, both $A^{(t)} = (a_{ij}^{(t)})_{1 \leq i,j \leq n}$ and $\widetilde{A}^{(t)} = (\widetilde{a}_{ij}^{(t)})_{1 \leq i,j \leq n}$ have zeros below their diagonal entries in the first $t-1$ columns; furthermore,

$$\text{if } \widetilde{a}_{it}^{(t)} = 0 \text{ for any } i \geq t \quad \Rightarrow \quad \widetilde{a}_{ht}^{(t)} = 0, \forall h \geq i. \qquad (2)$$

- The matrix $A^{(t+1)}$ is obtained from $\widetilde{A}^{(t)}$ making zeros in the column t below the main diagonal by adding an adequate multiple of the ith row to the $(i+1)$th, for $i = n-1, n-2, \ldots, t$, according to the following formula (for all $j \in \{1, 2, \ldots, n\}$)

$$a_{ij}^{(t+1)} = \begin{cases} \widetilde{a}_{ij}^{(t)}, & \text{if } 1 \leq i \leq t, \\ \widetilde{a}_{ij}^{(t)} - \dfrac{\widetilde{a}_{it}^{(t)}}{\widetilde{a}_{i-1,t}^{(t)}} \widetilde{a}_{i-1,j}^{(t)}, & \text{if } t+1 \leq i \leq n,\ \widetilde{a}_{i-1,t}^{(t)} \neq 0, \\ \widetilde{a}_{ij}^{(t)}, & \text{if } t+1 \leq i \leq n,\ \widetilde{a}_{i-1,t}^{(t)} = 0. \end{cases} \tag{3}$$

- Notice that in the end $A^{(n)} = \widetilde{A}^{(n)} = U$, and that when no row exchanges are needed, then $A^{(t)} = \widetilde{A}^{(t)}$ for all t.

- The element $p_{ij} = \widetilde{a}_{ij}^{(j)}$, $1 \leq j \leq i \leq n$, is called the (i,j) pivot of NE of A and the number

$$m_{ij} = \begin{cases} \dfrac{\widetilde{a}_{ij}^{(j)}}{\widetilde{a}_{i-1,j}^{(j)}} = \dfrac{p_{ij}}{p_{i-1,j}}, & \text{if } \widetilde{a}_{i-1,j}^{(j)} \neq 0, \\ 0, & \text{if } \widetilde{a}_{i-1,j}^{(j)} = 0\ (\Rightarrow \widetilde{a}_{ij}^{(j)} = 0), \end{cases} \tag{4}$$

the (i,j) multiplier.

- Matrices satisfying that NE can be performed Without changing Rows will be referred to as matrices verifying the *WR* condition.

- TP matrices verify the WR condition.

- A real matrix is called Totally Positive (TP) if all its minors are non-negative (Example: Bernstein basis, B-spline collocation matrices, etc.).

- A TP matrix A is said to be Almost Strictly Totally Positive (ASTP) if it satisfies the following condition: each minor C of A is positive if and only if all the diagonal elements of C are positive. (Example: B-spline collocation matrices, Hurwitz matrices, etc.)

- Now let us consider the important case in which NE can be performed without rows exchanges (*WR*). We denote by $E_i(\alpha)$, $2 \leq i \leq n$ the bidiagonal lower triangular matrix whose (r, s) entry $(1 \leq r, s \leq n)$ is given by

$$\begin{cases} 1, & \text{if } r = s, \\ \alpha, & \text{if } (r, s) = (i, i-1), \\ 0, & \text{otherwise.} \end{cases}$$

- Gasca and Peña has been proved that a nonsingular matrix of order n satisfies the *WR* condition if and only if it can be factorized in the form

$$A = [E_n(m_{n1})E_{n-1}(m_{n-1,1}) \cdots E_2(m_{21})] \cdots E_n(m_{n,n-1})U, \qquad (5)$$

with the $m'_{ij}s$ satisfying

$$m_{ij} = 0 \;\Rightarrow\; m_{hj} = 0, \; \forall\, h > i; \quad 1 \leq j \leq n-1, \; i \geq j+1.$$

- Example: NE of A column by column

$$\begin{pmatrix} \times & \times & \times & \times & \times \\ 4 & \times & \times & \times & \times \\ 3 & 7 & \times & \times & \times \\ 2 & 6 & 9 & \times & \times \\ 1 & 5 & 8 & 10 & \times \end{pmatrix}.$$

- However, as it was seen this factorization can be reordered in the following way

$$A = E_n(m_{n1}) \cdots [E_2(m_{21}) \cdots E_n(m_{n,n-1})]U,$$

which can also be written in the form

$$[E_n(-m_{n,n-1}) \cdots E_2(-m_{21})] \cdots E_n(-m_{n1})A = U.$$

- This means that NE of A can be done by subdiagonals, that is making zero first the $(n,1)$ entry, then the $(n-1,1)$ and $(n,2)$ ones, and so on, finishing making zeros the $(2,1)$, $(3,2)$, $\ldots$, $(n,n-1)$ entries.

- Example: NE of A by subdiagonals

$$\begin{pmatrix} \times & \times & \times & \times & \times \\ 7 & \times & \times & \times & \times \\ 4 & 8 & \times & \times & \times \\ 2 & 5 & 9 & \times & \times \\ 1 & 3 & 6 & 10 & \times \end{pmatrix}.$$

- We shall denote $F_i := E_{n-(i-1)}(m_{n-(i-1),1}) \cdots E_n(m_{ni})$,

$$F_i = \begin{pmatrix} 1 & & & & & & & & \\ 0 & 1 & & & & & & & \\ & \ddots & \ddots & & & & & & \\ & & 0 & 1 & & & & & \\ & & & m_{n-(i-1),1} & 1 & & & & \\ & & & & m_{n-(i-2),2} & 1 & & & \\ & & & & & \ddots & \ddots & & \\ & & & & & & m_{ni} & 1 \end{pmatrix}$$

and so the NE process (by subdiagonals) of a matrix A can be described in the following form

$$A^{[1]} := A,$$
$$F_t A^{[t+1]} := A^{[t]}, \quad 1 \leq t \leq n-1.$$

2.2 Pivoting strategies

- Now we present some pivoting strategies for NE, using row exchanges with similar purposes to those of the pivoting strategies for GE.

- Recall that $\widetilde{A}^{(t)}$ ($\widetilde{A}^{[t]}$ by subdiagonals) is obtained by a reordering of the rows of matrix $A^{(t)}$ ($A^{[t]}$ by subdiagonals) by an adequate pivoting strategy with a criterion for the choice of the pivots p_{ij}. Gaussian elimination with partial pivoting (GEPP) chooses the pivots so that all multipliers have absolute value not greater than 1.

- NE with partial pivoting: We interchange the rows of $A^{(t)}$ so that $\widetilde{A}^{(t)}$ satisfies $|\widetilde{a}_{tt}^{(t)}| \geq |\widetilde{a}_{t+1,t}^{(t)}| \geq \cdots \geq |\widetilde{a}_{nt}^{(t)}|$ and from $\widetilde{A}^{(t)}$ we construct $A^{(t+1)}$ as in (3).

- Pairwise pivoting for NE: To produce a zero at position (n, t) in $A^{(t)}$, one compares the element $a_{nt}^{(t)}$ with $a_{n-1,t}^{(t)}$. If $|a_{nt}^{(t)}| > |a_{n-1,t}^{(t)}|$ the corresponding rows are exchanged, in order that $|m_{nt}| \leq 1$. Then update the elements of row n. This process is repeated with $a_{it}^{(t)}$ and $a_{i-1,t}^{(t)}$, for $n-1 \geq i \geq t+1$.

- NE by subdiagonals with pairwise pivoting: If $|a_{n1}^{[1]}| > |a_{n-1,1}^{[1]}|$, then rows n and $n-1$ are interchanged, then row n is updated. The process is repeated with elements $a_{n-1,1}^{[2]}$ and $a_{n-2,1}^{[2]}$ and then a zero is made at position $(n-1, 1)$. The same procedure is performed on elements $a_{n2}^{[2]}$ and $a_{n-1,2}^{[3]}$ and so on.

2.3 Backward error analysis

- Let $Ax = b$ be a linear system where $A = LU$ is a nonsingular matrix, and assume that GE applied to A in floating point arithmetic has produced a factorization $\hat{L}\hat{U}$ and a solution $\hat{x}$. Wilkinson's componentwise backward error analysis of GE without pivoting was stated in the following way:

$$\hat{L}\hat{U} = A + E, \quad |E| \leq \gamma_n |\hat{L}|\,|\hat{U}|,$$

  where $\gamma_n := (nu)/(1 - nu)$ and u is the unit roundoff.

- If the matrix A is TP, de Boor and Pinkus showed that

$$|E| \leq \frac{\gamma_n}{1 - \gamma_{n-1}} A.$$

- If we consider now the equation $(A + H)\hat{x} = b$, they obtained that

$$|H| \leq 2\gamma_n |\hat{L}|\,|\hat{U}|, \quad |H| \leq \frac{2\gamma_n}{1 - \gamma_{n-1}} A.$$

- Other classes of matrices satisfy similar properties. For example, Higham proved that for some important classes of tridiagonal matrices A the backward error matrices E can also be chosen to be small componentwise relative to A.

- The next result, valid for the TP case, provides upper bounds for the matrices E and H.

Theorem 2.1

Let A be an $n \times n$ ($n \geq 2$) nonsingular TP matrix such that the computed matrices in the NE process satisfy that $\hat{A}^{[t]} \geq 0$, $\hat{F}_t \geq 0$, $\forall t$. For u sufficiently small the following properties hold:

1. *The matrices $\hat{L}$ and $\hat{U}$ computed by NE satisfy $\hat{L}\hat{U} = A + E$, with*

$$|E| \leq \gamma_{n-1} A. \tag{6}$$

2. *The solution of $Ax = b$ computed by NE, $\hat{x}$, satisfies $(A + H)\hat{x} = b$ where*

$$|H| \leq [\gamma_{n-1} + \gamma_n(1 + \gamma_{n-1})]A. \tag{7}$$

Contents

1. Introduction
2. Neville elimination
3. High Performance Computing
   - 3.1 Flops versus memory
   - 3.2 Blocking NE Algorithm for Exploiting Cache Memories
     - 3.2.1 Serial NE algorithm
     - 3.2.2 Partitioning NE Algorithm: 1D-blocked Algorithms
     - 3.2.3 2D-blocked Algorithms
   - 3.3 NE: Checkerboard Partitioning
     - 3.3.1 Algorithm
     - 3.3.2 Cost
     - 3.3.3 Efficiency
     - 3.3.4 Numerical results
4. References

- Parallel computing: the use of multiple computers or processors working together on a common task.
- Parallel computer: a computer that contains multiple processors
  - each processor works on its section of the problem,
  - processors are allowed to exchange information with other processors.

- Example: summing the elements of a vector.

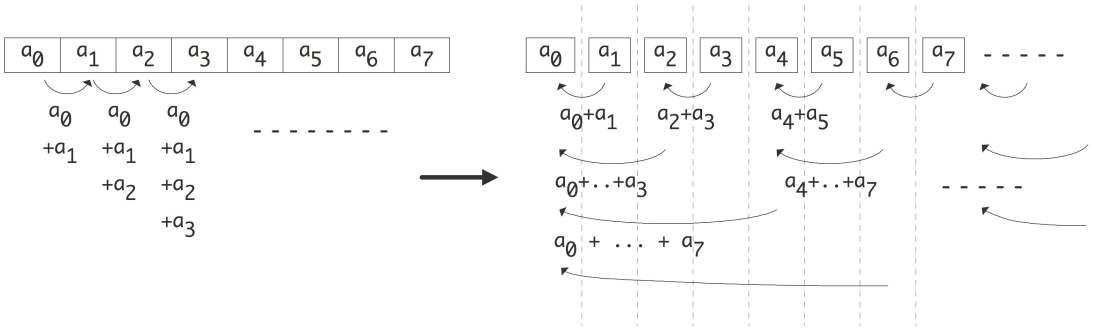


Figure 1:Parallelization of a vector addition.

- High Performance Computing (HPC) is the use of super computers and parallel processing techniques for solving complex computational problems.
- HPC is typically used for solving advanced problems and performing research activities through computer modeling, simulation and analysis.
- Many problems have been solved with HPC techniques that were impossible to solve with individual workstations or personal computers.

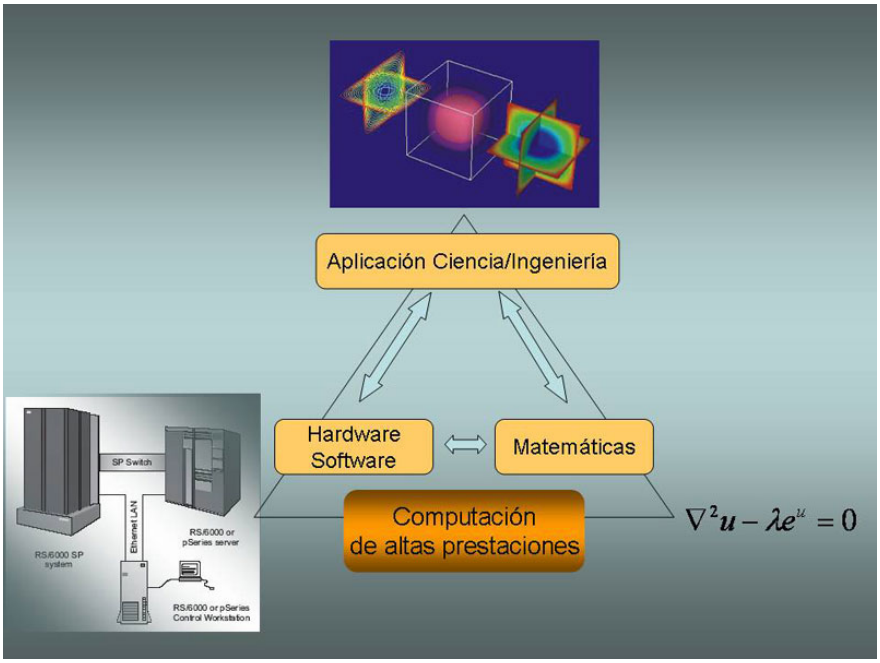


Figure 2: Action scheme.

- The recent availability of advanced-architecture computers has had a significant impact on all spheres of scientific computation, including algorithm research and software development in numerical linear algebra.
- Some researchers have used similar ideas to NE when looking for efficient pivoting strategies of elimination procedures to solve linear systems in parallel computing.
- Other researchers focus their works on the block implementation to improve the performance of some algorithms.
- To understand how these algorithms work, it is necessary to understand how the time is spent by the computer while executing and how computer memories operate.

3.1 Flops versus memory

- Computer memory is built as a hierarchy, with a series of different kinds of memories ranging from very fast at the top of the hierarchy down to slow at the bottom.
- The fast memory is expensive and small while slow memory is cheaper and larger.
- Useful operations can be done only on data at the top of the hierarchy. Thus, data at one level of the memory hierarchy can move to adjacent levels.
- This means that an ill-designed algorithm may spend most of its time moving data from the bottom of the hierarchy to the top.

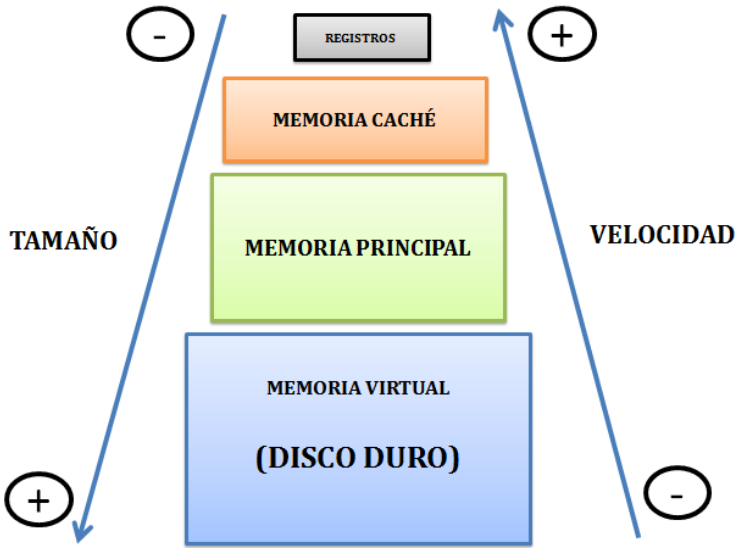


Figure 3: Memory hierarchy.

- Several ways of improving the performance of an algorithm can be taken into account. Three remarkable issues are the use of parallel computing, the profitable usage of memory hierarchies and the work with matrix operations.
- There exist algorithms that operate on square or rectangular sub-blocks of matrices in their innermost loops rather than on entire rows or columns with the goal of improving their performance.
- For example, some authors reorganizes Gaussian elimination to use Level-3 BLAS (Basic Linear Algebra Subroutines) operations.

- Table 1 shows the magnitudes associated with parameters f, m and q for different levels of BLAS:
  - f: numbers of floating point operations executed,
  - m: numbers of access to memory,
  - q: how many floating point operations we can perform on average per slow memory reference (how much useful work we can do compared to the time moving data).

Operation	Definition	f	m	q=f/m
BLAS 1				
`saxpy`	$y \leftarrow \alpha \cdot x + y$	$2n$	$3n+1$	$2/3$
BLAS 2				
`Matrix-vector mult`	$y \leftarrow A \cdot x + y$	$2n^2$	n^2+3n	2
BLAS 3				
`Matrix-matriz mult`	$C \leftarrow A \cdot B + C$	$2n^3$	$4n^2$	$n/2$

Table 1: Flops versus memory.

- For example, the number of elementary operations to calculate $y_i \leftarrow \alpha \cdot x_i + y_i$, with $i = 1, 2, \ldots, n$, es $2n$, since in each step it does 1 addition and 1 product.
- The number of memory accesses (m) needed to implement `saxpy` is $3n + 1$,
  - read n values de x_i, n values of y_i and one value for α from memory "slow" to memory "fast" ($2n + 1$),
  - next, write n values of y_i (n).
- Finally, the value obtained to q is $2/3$.

3.2 Blocking NE Algorithm for Exploiting Cache Memories

- First, we explore block algorithms for NE which take into account the memory hierarchies of a computer. These algorithms try to manage the memory movements to optimize them.
- Thus, the matrix of the system is divided following three different strategies:
  - 1D-blocked: rows, columns.
  - 2D-blocked.

3.2.1 Serial NE algorithm

Algorithm 1 A serial NE algorithm

```
1: for k = 1 : n − 1 do
2:     for i = n : −1 : k + 1 do
3:         A(i, k) = A(i, k)/A(i − 1, k)
4:         for j = k + 1 : n do
5:             A(i, j) = A(i, j) − A(i, k)A(i − 1, j)
6:         end for
7:     end for
8: end for
```


- Let us analyze the parameter q for NE of Algorithm 1.
- With regard to the floating point operations, NE without back substitution performs

$$f = \frac{2n^3}{3} - \frac{n^2}{2} - \frac{n}{6} \simeq \frac{2}{3}n^3 \quad \text{flops.} \tag{8}$$

- For each step k, NE uses $n - k + 1$ rows and $n - k$ columns so that $(n - k + 1)(n - k)$ elements are being read, keeping them in fast memory until it is no longer needed and then moving it back to slow memory.

- If the cache memory can hold the whole matrix then $q = \Theta(n)$. But in general, the size of cache memory is bigger than a few elements.
- Let us consider the case where $2n < M \ll n^2$; this means that the fast memory is large enough to hold two matrix columns or rows.
- Then, we can improve the performance of Algorithm 1 by making as much zeros as possible in the part of the matrix which is currently at fast memory.
- Therefore, once a row is read into fast memory, it is kept into it until it is no possible to make more zeros over it.

- From Table 1, we can deduce the number of read ($n^3/3$) and written elements ($n^3/3$).

Step k	Read into fast memory	Write back into slow memory
i=n	rows n and $n-1$	row n
i=n-1	row $n-2$	row $n-1$
.	.	.
.	.	.
.	.	.
i=k+1	row k	row $k+1$

Tabla 2: Read and written rows at each iteration.

- With the use of this strategy for moving elements from fast memory to slow memory (and viceversa), the total number of memory accesses is

$$m \simeq \frac{2}{3}n^3. \tag{9}$$

- Then, from equations (8) and (9) it is concluded that q takes the value

$$q \simeq 1. \tag{10}$$

3.2.2 Partitioning NE Algorithm: 1D-blocked Algorithms

- In this case the matrix is divided into p row blocks of r rows ($pr = n$), making zeros below the diagonal in all the columns where we can do it.
- Therefore, once a row block is moved to fast memory, we make as many zeros as possible. Algorithm 2 shows this procedure and Figure 4 is an example of this strategy.
- As it can be seen in Algorithm 2, in this case Neville algorithm is performed in $p - 1$ steps. Thus, at each step i, $i + 1$ blocks must be read to fast memory and come back to slow memory.

Algorithm 2 NE when data are partitioned by rows

1: **for** $i = 1 : p - 1$ **do**
2: {Read B_{p-i} into fast memory}
3: **for** $k = 1 : i$ **do**
4: {Read B_{p-i+k} into fast memory}
5: if $(k == i)$ then $\omega = (k+1)r - 1$ else $\omega = kr$ end if
6: **for** $j = (k-1)r + 1 : \omega$ **do**
7: Compute the multipliers in the active part of column j of B_{p-i+k} and $B_{p-i+k-1}$
8: Update the active part of B_{p-i+k} and $B_{p-i+k-1}$
9: **end for**
10: {Write back $B_{p-i+k-1}$ into slow memory}
11: **end for**
12: {Write back B_p into slow memory}
13: **end for**

- Then the final number of moved blocks is approximately p^2.
- As each block has nr elements, the number of moved elements is

$$nrp^2 = pn^2. \tag{11}$$

- Then, q is

$$q = \frac{f}{m} = \frac{\frac{2n^3}{3}}{pn^2} = \frac{2r}{3}. \tag{12}$$

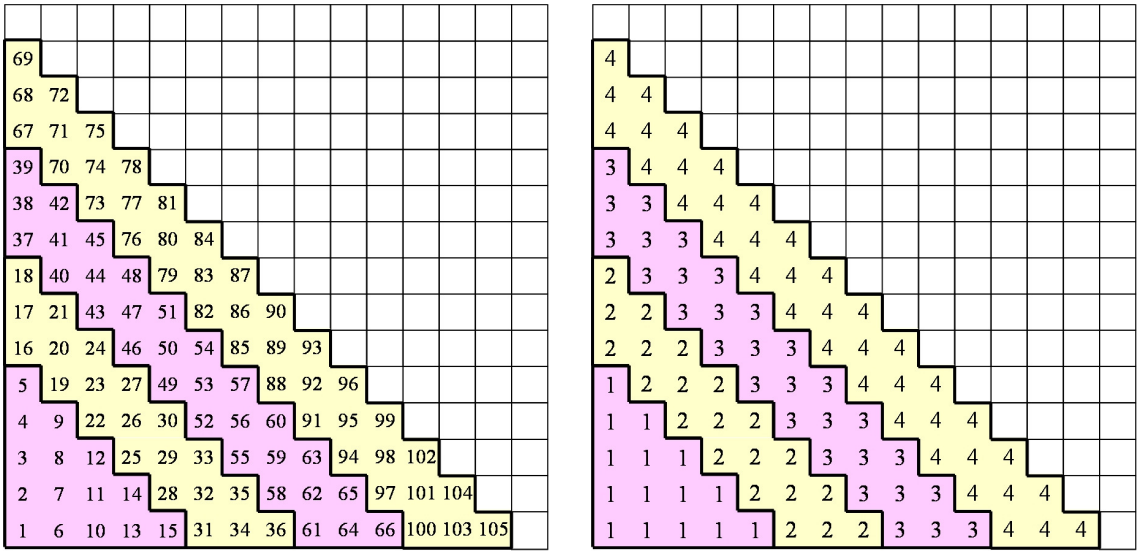


Figura 4: Example ($n = 15$, $p = 5$, $r = 3$) of the way the zeros are made according to the row blocked strategy.

- Now, the matrix is divided into p column blocks of r columns ($pr = n$). Taking into account that once a block is moved to fast memory, we make as much zeros as possible.
- Algorithm 3 shows NE in this case and Figure 5 shows the order in which the zeros are made.
- According to Algorithm 3 the number of blocks moved is approximately $p^2/2$. Thus, the elements moved are $pn^2/2$. In this case q is

$$q = \frac{f}{m} = \frac{\frac{2n^3}{3}}{\frac{pn^2}{2}} = \frac{4r}{3}.$$

Algorithm 3 NE when data are partitioned by columns

1: **for** $i = 1 : p$ **do**
2: {Read B_i into fast memory}
3: **for** $j = 1 : i - 1$ **do**
4: {Read B_j into fast memory}
5: Update the active part of B_i with the multipliers stored on B_j
6: **end for**
7: if $(i == p)$ then $\omega = n - 1$ else $\omega = ir$ end if
8: **for** $j = (i - 1)r + 1 : \omega$ **do**
9: Compute the multipliers in the active part of column j of B_i
10: Update the active part of B_i
11: **end for**
12: {Write back B_i into slow memory}
13: **end for**

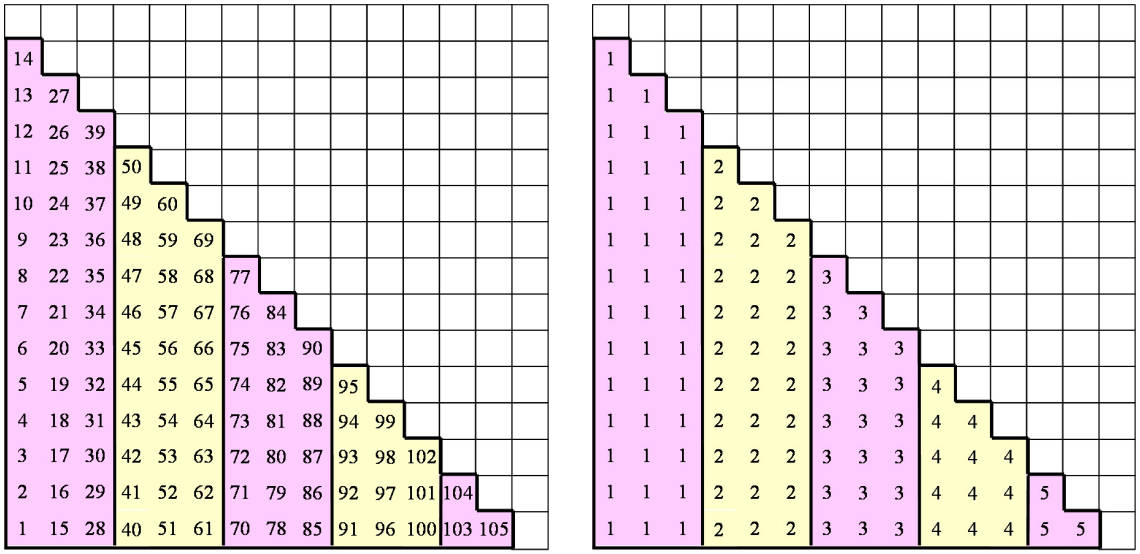


Figura 5: Example ($n = 15$, $p = 5$, $r = 3$) of the way the zeros are made according to the column blocked strategy.

3.2.3 2D-blocked Algorithms

- In this case the matrix is divided into p^2 blocks of order $r = n/p$. So when a block is moved to fast memory, we make as much zeros as possible. According to Algorithm 4 the number of moved blocks is

$$\sum_{i=2}^{p}\sum_{j=1}^{p}(i+1) + 2\sum_{i=1}^{p}(p-i+1) = \frac{p^3}{2} + \frac{5p^2}{2} - p \approx \frac{p^3}{2}.$$

Algorithm 4 NE when data are partitioned by submatrices

```
1: for i = 1 : p do
2:    if i > 1 then
3:       Part I
4:    end if
5:    {Read B_pi into fast memory}
6:    if i < p then
7:       Part II
8:    else
9:       Part III
10:   end if
11:   {Write back B_ii into slow memory}
12: end for
```

Part I:

for $j = p : -1 : 1$ **do**
  {Read B_{ji} into fast memory}
  for $k = 1$ to $i - 1$ do
    {Read B_{jk} into fast memory}
    Update the active part of B_{ji} with the multipliers stored on B_{jk}
  end for
  {Write back B_{ji} into slow memory}
end for

Part II:

for $j = p - 1 : -1 : i$ **do**
  {Read B_{ji} into fast memory}
  for $k = \frac{n}{p}(i-1) + 1 : \frac{n}{p} i$ **do**
    Compute the multipliers in the active part of column k
    of $B_{j+1,i}$ and B_{ji}
    Update the active parts of $B_{j+1,i}$ and B_{ji}
  end for
  {Write back $B_{j+1,i}$ into slow memory}
end for

Part III:

for $k = \frac{n}{p}(i-1) + 1 : n - 1$ **do**
  Compute the multipliers in the active part of column k of B_{ii}
  Update the active parts of B_{ii}
end for

- As the size of the fast memory M has to be at least $2r^2$ (because we need two blocks in fast memory to perform the algorithm) r is approximately $\sqrt{M/2}$ and then

$$q = \frac{4}{3}\sqrt{\frac{M}{2}},$$

  which means that q only depends on the size of fast memory.

- In particular q grows independently of n as M grows, which means that we expect the algorithm to be fast for any matriz size n and go faster if the fast memory size M is increased. There are both attractive properties.

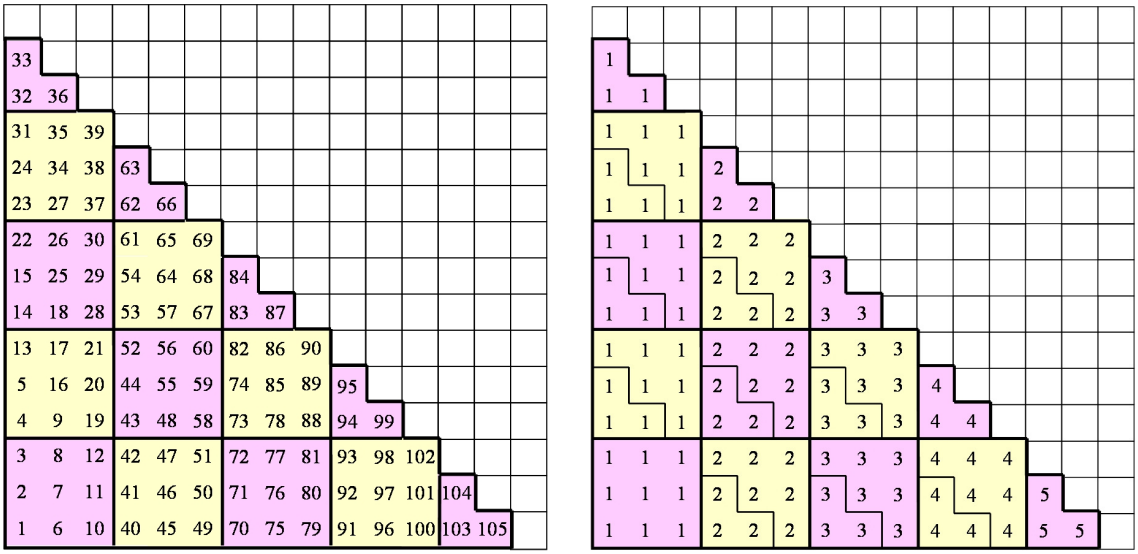


Figure 6: Example ($n = 15$, $p = 5$, $r = 3$) of the way the zeros are made according to the 2D-blocked strategy.

3.3 NE: Checkerboard Partitioning

- To process a matrix in parallel, we must partition it so that the partitions can be assigned to different processors. The data partitioning significantly affects the performance of a parallel system. Hence, it is important to determine which data-mapping scheme is the most appropriate one for each algorithm.
- In order to distribute the matrices over the processors, two different ways are usually taken into account: the striped partitioning and the checkerboard partitioning.
- When a striped partitioning is performed, the matrix is divided into groups of complete rows or columns. In this case, each processor is assigned to one such group.

- In a checkerboard partitioning, the matrix is divided into smaller square or rectangular blocks of submatrices that are distributed among processors.
- It is well known that checkerboard partitioning can exploit more concurrency than striped partitioning because the matrix computation can be divided among more processors than in the case of striping.
- Next, we analyze the performance of Neville method when a checkerboard partitioning is performed.

3.3.1 Algorithm

- We consider a parallel implementation of Neville algorithm in which the matrix A of the linear equation system $Ax = b$ is divided into smaller square blocks.
- Consider a two-dimensional wraparound mesh of k processors, such that $k \leq n^2$.

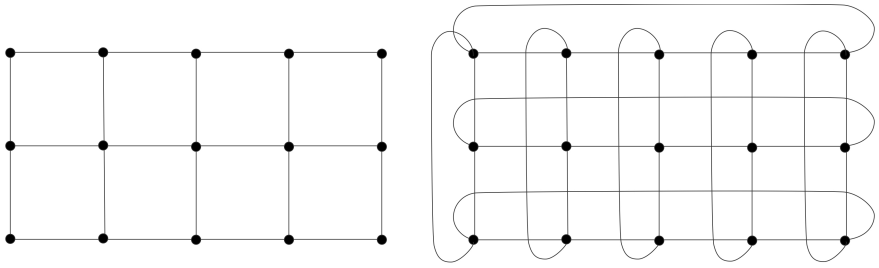
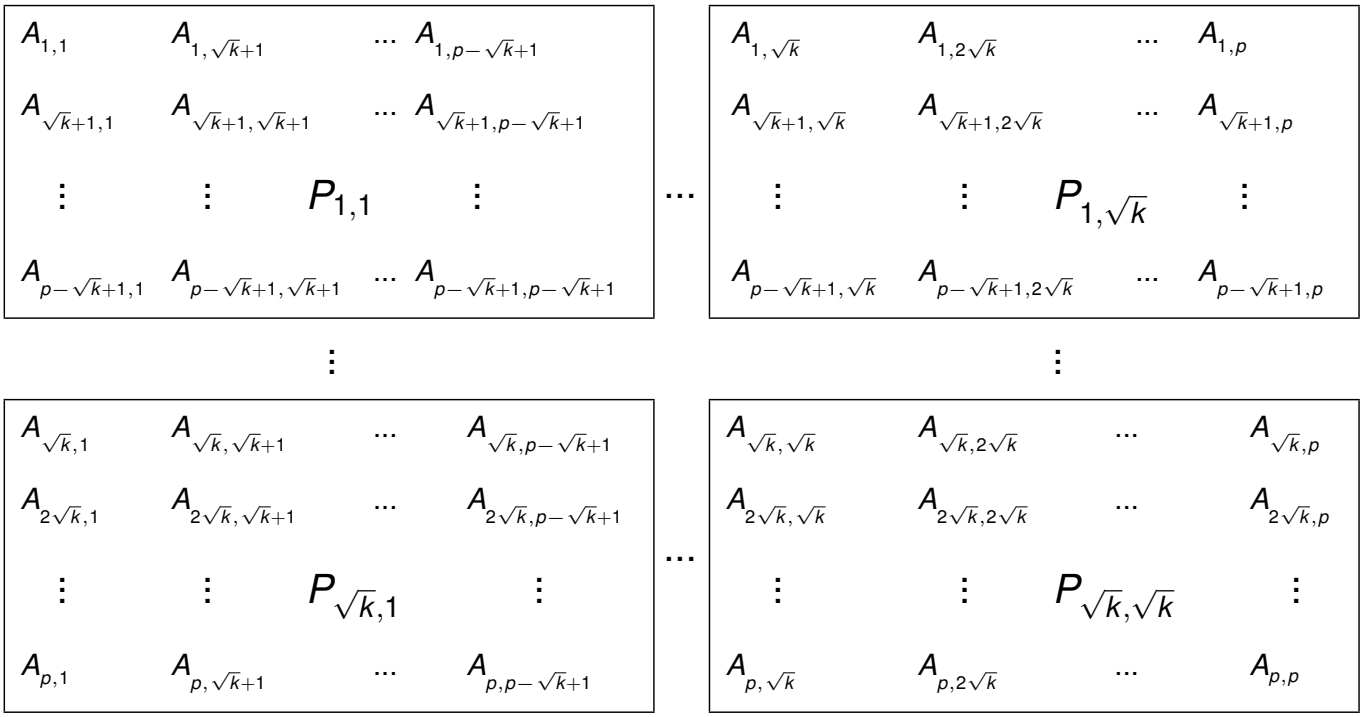


Figure 7: A 2D grid with torus connections.

- The processor in row i and column j is denoted by $P_{i,j}$, with $i = 1, ..., \sqrt{k}$ and $j = 1, ..., \sqrt{k}$.
- A two-way communication line links $P_{i,j}$ to its neighbors $P_{i+1,j}$, $P_{i-1,j}$, $P_{i,j+1}$ and $P_{i,j-1}$.
- Let $A = (A_{i,j})_{1 \leq i,j \leq p}$ be an $n \times n$ matrix, $A_{i,j}$ a $m \times m$ submatrix of A for $1 \leq i, j \leq p$, then each one of the $A_{i,j}$ blocks are split among the processors in a cyclic way as it is shown in next diagram:



- This partitioning, called block-cyclic-checkerboard, assigns $h \times h$ blocks of $m \times m$ consecutive elements to each processor, being $h = n/(m\sqrt{k})$ with $m \in [1, n/\sqrt{k}]$.
- If $m = n/\sqrt{k}$ this algorithm is known as block-checkerboard partitioning.
- If $m = 1$ the method is named cyclic-checkerboard partitioning.
- We focused our study in the more general case (and more efficient case): block-cyclic-checkerboard.

- Let us study the parallel algorithm performance in an iteration j. In this iteration the variable x_j must be removed.
- In order to get this result it is necessary to follow four steps:
  - Step 1. A processor $P_{i,l}$ sends the last row of each block to the processor $P_{i+1,l}$ when the index of such row is larger than j. If $i = \sqrt{k}$ the elements are transferred to processor $P_{1,l}$. These rows are necessary to obtain $A^{(j+1)}$ from $A^{(j)}$.
  - Step 2. Compute the multipliers to get zeros in entries (n, j), $(n-1, j)$, $\ldots$, $(j+1, j)$.
  - Step 3. Transfer the multipliers computed in $P_{i,l}$ (in *Step 2*) to each active processor being in the row i of the mesh.
  - Step 4. Actualize the elements $a_{i,t}^{(j+1)}$.

3.3.2 Cost

- Let $T(n; k)$ be the total parallel cost when k processors are used. This cost is obtained by computing communication and computation time. Hence

$$T(n; k) = t_{\text{computation}} + t_{\text{communication}}.$$

- Computation Time: Hence, computation cost is studied for processor $P_{\sqrt{k},\sqrt{k}}$ because it is the processor that makes the largest number of computations.

$$t_{\text{computation}} \approx \left(4n^2 + 6mn\sqrt{k} - 6mn + 2m^2k - 2m^2\sqrt{k}\right)\frac{n}{6k}t_c.$$

being t_c the time for making a float operation.

- Communication Time: We consider an approximation based on a MIMD (multiple instruction, multiple data) distributed memory architecture, where the processors are connected by an interconnection network.

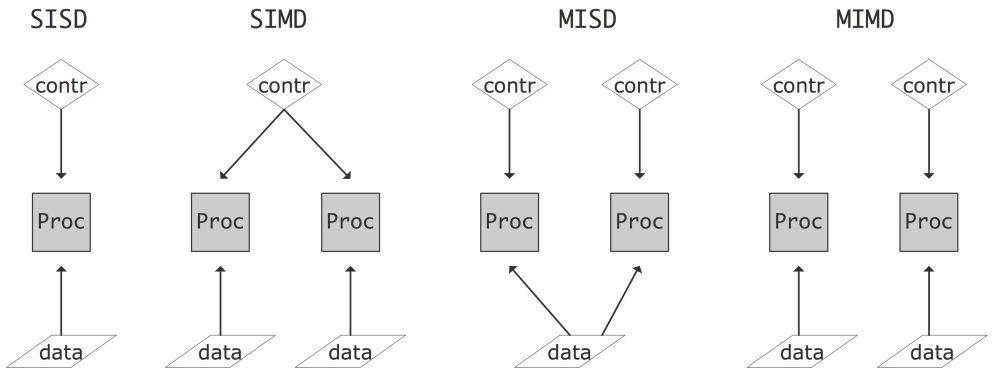


Figure 8: The four classes of the Flynn taxonomy.

- In this case, a message containing α words (float numbers) can be sent between directly-connected processors in time

$$t_{\text{communication}}(\textit{direc_connec}) = t_s + \alpha t_w,$$

where t_s is the time required to handle a message at a sending processor and t_w is the transfer time of a number in float point.

- The communication time of one-to-all broadcast with a cut-through routing is

$$t_{\text{communication}}(\textit{one_to_all}) = \log_2(k)t_s + \alpha \log_2(k)t_w.$$

- In the Step 1 a processor $P_{i,l}$ sends the last row of each block to processor $P_{i+1,l}$, so a communication between neighbor processors is made. The processor which makes the maximum number of communications is $P_{\sqrt{k}-1,\sqrt{k}}$.
- On the other hand, in Step 3 a processor transfers the multipliers to the processors that need them (one-to-all broadcast). The processors of the row $\sqrt{k}$ make the maximum number of communications.
- Therefore the communication time ($t_{communication}$) is approximately

$$n\log_2(\sqrt{k})t_s + \left(\frac{n^3}{3mk} + \frac{n^2}{2\sqrt{k}} + \frac{nm}{6} + \frac{n^2\log_2(\sqrt{k})}{2\sqrt{k}} + \frac{nm\log_2(\sqrt{k})}{2}\right)t_w.$$

3.3.3 Efficiency

- The performance of this partitioning in NE is studied by measuring its efficiency (being k the number of processors):

$$E = \frac{T(n;1)}{k\ T(n;k)}.$$

- The theoretic efficiency reached for large values of n is shown in next Table:

Distribution	*Efficiency*
block-checkerboard	$\approx \frac{\sqrt{k}}{3\sqrt{k}-2}$
cyclic-checkerboard	$\approx \frac{2t_c}{2t_c+t_w}$
block-cyclic-checkerboard	$\approx \frac{2t_c}{2t_c+\frac{1}{m}t_w}$

3.3.4 Numerical results

- The theoretic model's constants used in this work have been obtained by using a least squares approximation.
- Parallel PC cluster using public domain Message Passing Interface (MPICH) and Linux Terminal Sever Project (LTSP):

  $t_c = 3.8 * 10 - 8\ sec.,\ t_s = 8.8 * 10 - 5\ sec.,\ t_w = 1.2 * 10 - 6\ sec.$
- IBM SP2 (IBM SCALABLE POWER PARALLEL SYSTEM 2):

  $t_c = 1.6 * 10 - 8\ sec.,\ t_s = 3.1 * 10 - 5\ sec.,\ t_w = 1.4 * 10 - 7\ sec.$

- The maximum efficiency is obtained by using an adequate block size, called optimum block size:

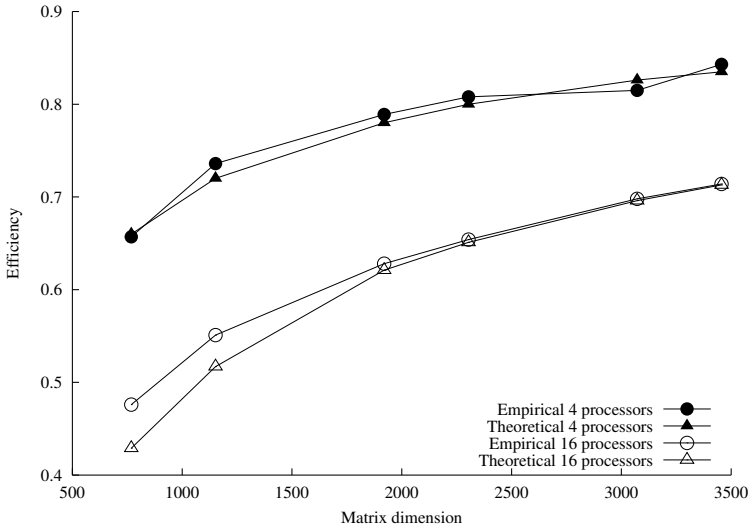


Figure 9: Efficiency for block-cyclic-checkerboard partitioning.

- As the performance of both efficiencies (theoretical and empirical) is quite similar we can extrapolate the behavior of the empirical model using the theoretical one:

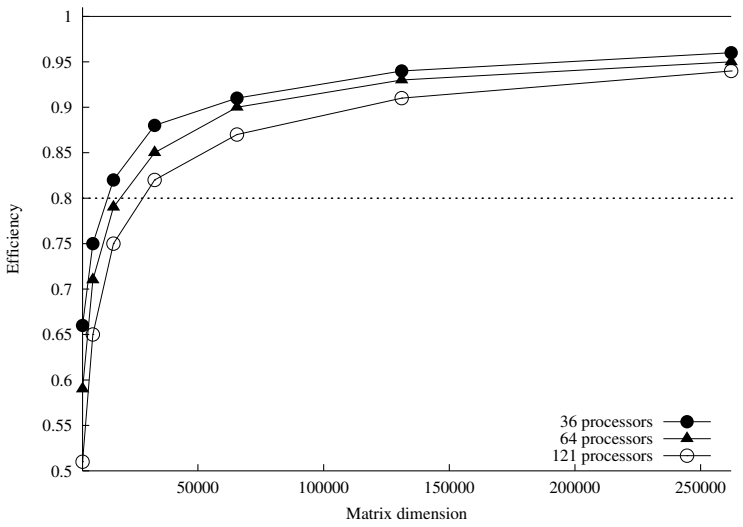


Figure 10: Estimated efficiency for block-cyclic-checkerboard partitioning.

Contents

1 Introduction

2 Neville elimination

3 High Performance Computing

4 References

4 References

- F. Almeida, D. Giménez, J.M. Mantas, A.M. Vidal, *Introducción a la programación paralela*, Paraninfo Cengage Learning, 2008.
- P. Alonso, R. Cortina, I. Díaz, J. Ranilla, *Blocking NE Algorithm for Exploiting Cache Memories*, Applied Mathematics and Computation, 209 (2009), pp. 2-9.
- P. Alonso, R. Cortina, I. Díaz, J. Ranilla, *Neville Elimination: A study of the efficiency using Checkerboard Partitioning*, Linear Algebra and its Applications, 393 (2004), pp. 3-14.
- P. Alonso, R. Cortina, E.S. Quintana-Ortí, J. Ranilla, *A Increasing Data Locality and Introducing Level-3 BLAS in the Neville Elimination*, Applied Mathematics and Computation, 218(7) (2011), pp. 3348-3358.

- P. Alonso, M. Gasca, J.M. Peña, *Backward Error Analysis of Neville Elimination*, Applied Numerical Mathematics, 193 (1997), pp. 193-204.
- J.W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.
- M. Gasca, J.M. Peña, *Total positivity and Neville elimination*, Linear Algebra and its Applications, 165 (1992), pp. 25-44.
- G.H. Golub, C.F. Van Loan, *Matrix computations* (3rd ed.), Johns Hopkins University Press, 1996.
- A. Grama, G. Karypis, V. Kumar, A. Gupta, *Introduction to Parallel Computing* (2nd ed.), Addison-Wesley, 2003.

Thank you very much