CASOS DE ESTUDIO

Máster de Modelización e Investigación Matemática, Estadística y Computación

Programación científica

Parte IV

Curso 2014/2015

Contenido

Contenido

1.1 Caso 1: Blocking Neville Elimination Algorithm for Exploiting Cache Memories

- In this work we explore block algorithms for Neville elimination (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, blocking by rows, columns or submatrices. In each case we study the performance of the algorithm according to the ratio of floating point operations to memory references (q).

  P. Alonso, R. Cortina, I. Díaz, J. Ranilla,
  Blocking NE Algorithm for Exploiting Cache Memories,
  Applied Mathematics and Computation, 209 (2009), pp. 2-9

1.1.1 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 or Computer Aided Geometric Design (CAGD).

Description

- NE is a procedure to make zeros in a column of a matrix adding to each row a multiple of the previous one.

- 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:

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

  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,

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

- In order to get (2) the matrix $\widetilde{A}^{(t)}$ is obtained from $A^{(t)}$ by moving to the bottom the rows with a zero entry in column t, if necessary.
- 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 i-th 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.

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
```


- Matrices satisfying that NE can be performed Without changing Rows will be referred to as matrices verifying the *WR* condition.
- A real matrix is called TP if all its minors are nonnegative (Example: Bernstein basis, B-spline collocation matrices, etc.).
- A TP matrix A is said to be 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.)
- TP matrices verify the WR condition.

1.1.2 Serial NE algorithm

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

$$f = \frac{2n^3}{3} - \frac{n^2}{2} - \frac{n}{6} \simeq \frac{2}{3}n^3 \tag{5}$$

floating point operations.

- If the cache memory can hold the whole matrix then $q = \Theta(n)$. However, if the size of fast memory is too small (it contains few elements) the total number of reading operations is

$$\sum_{k=1}^{n-1} \sum_{i=k+1}^{n} 2(n-k+1) \simeq \frac{2}{3}n^3 \tag{6}$$

and the total number of writing operations is

$$\sum_{k=1}^{n-1} \sum_{i=k+1}^{n} (n-k) \simeq \frac{1}{3}n^3. \tag{7}$$

- Therefore, from expressions (6) and (7) the number to memory accesses m comes to $m \simeq n^3$. Then,

$$q \simeq \frac{2}{3}. \tag{8}$$

- 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 1: 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 (5) and (9) it is concluded that q takes the value

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

1.1.3 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 1 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

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

- The fast memory must contain two blocks at the same time to perform the algorithm. Therefore its size must be at least $2rn$, which implies $r \simeq \frac{M}{2n}$. Replacing r by this last approximation in (12) we obtain

$$q \simeq \frac{M}{3n}. \tag{13}$$

- In this case, the lowest performance (with regard to parameter q) is got when the size of M is $2n$, namely, when the data are moved from slow memory to fast memory row by row.
- On the other hand, when the size of M is $2n^2$ it is obtained the highest q.
- Therefore, $r = n$ and $p = 1$ which means the whole matrix is moved at the same time. This is obviously a too expensive memory requirement. Thus, there exists a trade-off between the size of the fast memory and q.

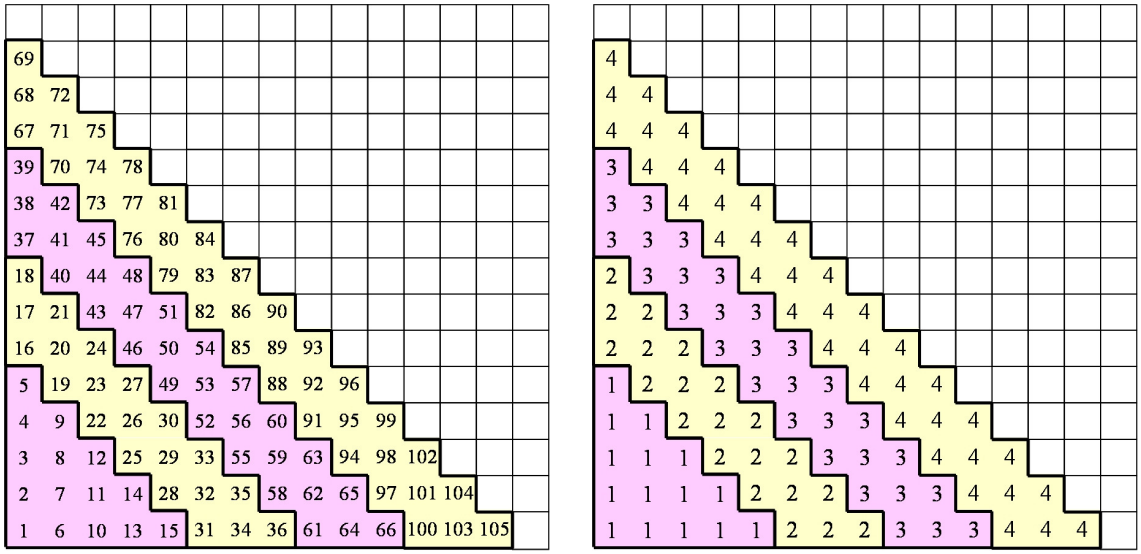

Figura 1: Example ($n = 15$, $p = 5$, $r = 3$) of the way the zeros are made according to the row blocked strategy.

- 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 4 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

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

- As in case of row blocking, the lowest q is got when $r = 1$ and the highest when $r = n$. In addition, q is higher than the obtained for Level 1 BLAS operations for all r.

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**

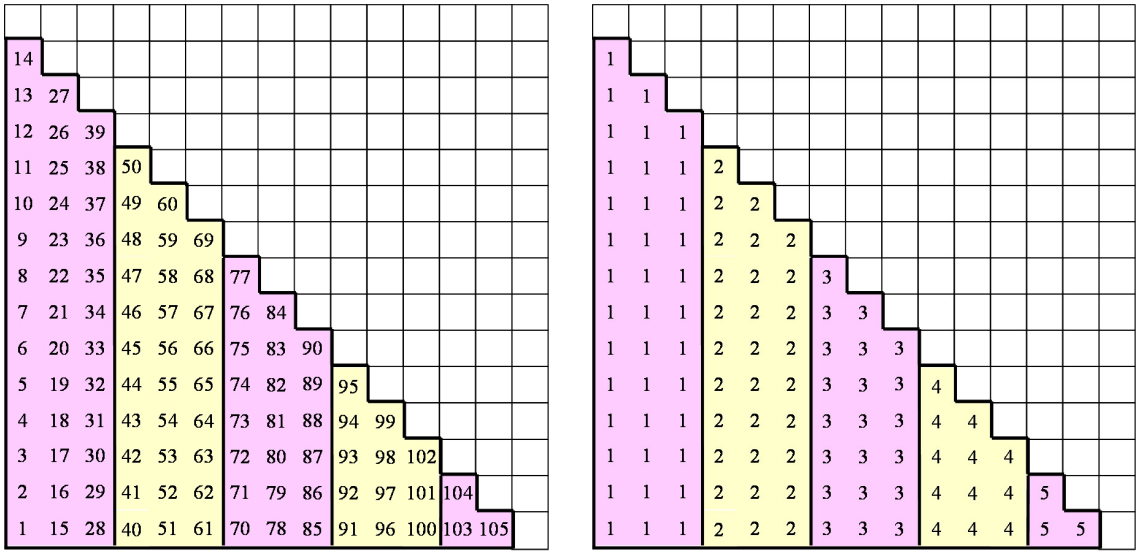

Figura 2: Example ($n = 15$, $p = 5$, $r = 3$) of the way the zeros are made according to the column blocked strategy.

1.1.4 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. Algorithm 4 shows this case. According to Algorithm 4 the number of moved blocks is

$$\sum_{i=2}^{p}\sum_{j=1}^{p}(i+1)+2\sum_{i=1}^{p}(p-i+1)=\frac{p^3}{2}+\frac{5p^2}{2}-p\approx\frac{p^3}{2}. \qquad (14)$$

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.

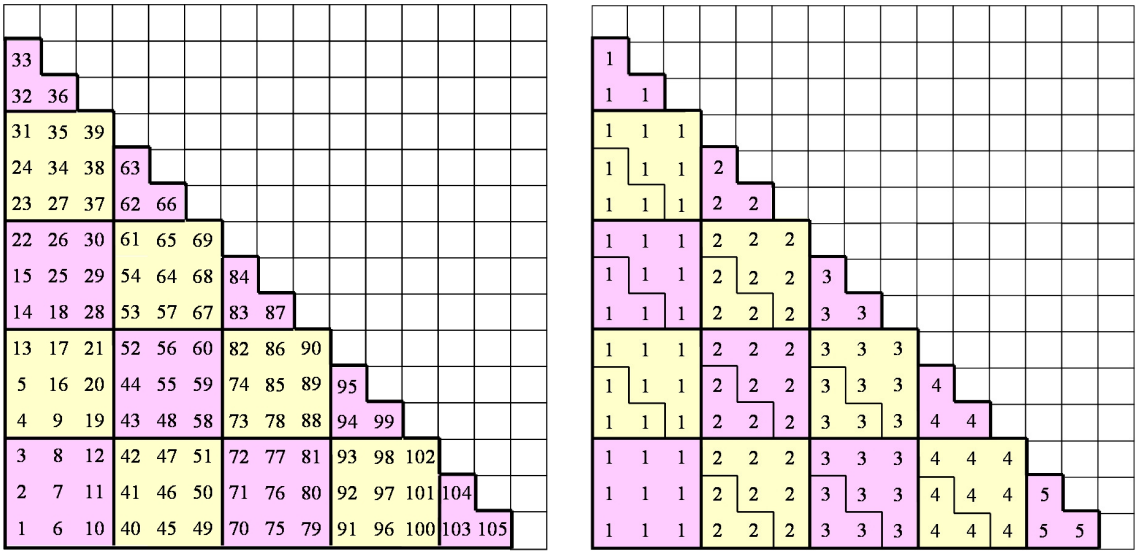

Figura 3: Example ($n = 15$, $p = 5$, $r = 3$) of the way the zeros are made according to the 2D-blocked strategy.

Remarks:

- With regard to $1D$-algorithms we can see that q is dependent not only on the cache memory size but also on the matrix size. Thus, $1D$-algorithms do not reach a good performance due to their lack of scalability.
- On the other hand, the performance obtained for $2D$-algorithms depends on the size of fast memory. Thus, an increase on M produces an increase on q so that it is possible to overtake the value of $n/2$ got when matrix operations are considered. Nevertheless, it is easy to obtain a performance between Level-2 and Level-3 BLAS if the fast memory has a reasonable size.
- Therefore, the best strategy is to use $2D$-algorithms, that is, partitioning the coefficient matrix by submatrices.

Contenido

1.2 Caso 2: Detecting point sources in CMB maps using an efficient parallel algorithm

- The Cosmic Microwave Background (CMB) is a diffuse radiation which is contaminated by the radiation emitted by point sources. The precise knowledge of CMB fluctuations can lead to a better knowledge of the chemistry at the early stages of the Universe.
- In this work, we present an efficient algorithm, with a high degree of parallelism, which can improve, from the computational point of view, the classical approaches for detecting point sources in Cosmic Microwave Background maps.

- High performance computing libraries and parallel computing techniques have allowed to construct a portable, fast and numerically stable algorithm.
- To check the performance of the new method, we have carried out several simulations resembling the observational data collected by the Low Frequency Instrument of the Planck satellite. The sources are detected in their real positions.

  P. Alonso, F. Argüeso, R. Cortina, J. Ranilla, A. M. Vidal,
  Detecting Point Sources in CMB Maps using an Eficient Parallel Algorithm,
  J. Math. Chem. 50(2012), pp. 410-420.

1.2.1 Problem description

- In a region of the celestial sphere, we suppose to have a certain number n of radio sources.
- The emission of these sources is superimposed to the radiation $f(x,y)$. In our particular case this radiation is the CMB. A model for the emission as a function of the position (x,y) is:

$$\tilde{d}(x,y) = f(x,y) + \sum_{\alpha=1}^{n} a_\alpha \, \delta(x - x_\alpha, y - y_\alpha)$$

  where $\delta(x,y)$ is the 2D Dirac delta function, the pairs are the locations of the point sources in our region of the celestial sphere, and a_α are their intensities.

- We observe this radiation through an instrument, with beam pattern $b(x,y)$, and a sensor that adds a random noise $n(x,y)$ to the signal measured.
- Again, as a function of the position, the output of our instrument is:

$$d(x,y) = \sum_{\alpha=1}^{n} a_\alpha \; b(x-x_\alpha, y-y_\alpha) + (f * b)(x,y) + n(x,y) \qquad (15)$$

  where the point sources and the diffuse radiation have been convolved with the beam.
- In our application, we are interested in extracting the locations and the intensities of the point sources.

- We assume that the intensities of the point sources are sufficiently above the level of the rest of the signal. If $c(x,y)$ is the signal which does not come from the point sources and $b(x,y)$ is the beam pattern, we have:

$$d(x,y) = \sum_{\alpha=1}^{n} a_\alpha \; b(x-x_\alpha, y-y_\alpha) + c(x,y). \qquad (16)$$

- If our data set is a discrete map of N pixels, the above equation can be rewritten in vector form where d and c are the lexicographically ordered versions of the discrete map $d(x,y)$ and $c(x,y)$ respectively, a is the n-vector containing the positive source intensities a_α and ϕ is an $N \times n$ matrix whose columns are the lexicographically ordered versions of n replicas of the map $b(x,y)$. Equation (16) becomes

$$d = \phi \; a + c. \qquad (17)$$

- Looking at Eqs. (16) and (17), once n and (x_a, y_a) are known, matrix ϕ is perfectly determined. Let us denote the list of source locations by the $n \times 2$ matrix R, containing all their coordinates. For the CMB we can assume that c is a Gaussian random field with zero mean and known covariance ξ. Thus the likelihood function is

$$p(d|n, R, a) = \exp(-(d - \phi\ a)^t \xi^{-1} (d - \phi\ a)/2). \qquad (18)$$

- If we define $M = \phi^t \xi^{-1} \phi$, $e = \phi^t \xi^{-1}\ d$, the maximization of (18) leads us to the linear system

$$Ma = e \qquad (19)$$

  where M is a $n \times n$ matrix. The solution of this system will yield the maximum likelihood estimator of the source intensities.

- In principle, we know neither the number n of point sources nor their positions. One standard way of dealing is considering (19) the local maxima of e and selecting as source positions these local maxima above a certain threshold.
- Finally, we have to find suitable methods to solve the system shown in (19), taking into account that the number of sources can range from several to thousands depending on the size of the region studied and also on the frequency analyzed.
- We provide an efficient solution with both aspects in mind: to get a reasonable run time and a numerically stable algorithm.

1.2.2 Efficient implementations

- It should be noted that when we are building the system $Ma = e$ we consider that the matrices M, ϕ and ξ are of order N, while the vectors e and d have N rows.
- Once vector e has been filtered by using the fixed threshold, matrix ϕ is set to $N \times n$ by choosing the adequate rows. The matrices ϕ and ξ can be generated from: the number of pixels (N), the pixel size (PIX) and the full width half maximum of the beam ($FWHM$).
- To solve the system $Ma = e$ is necessary to calculate the matrix $M = \phi^t \xi^{-1} \phi$, and the vector $e = \phi^t \xi^{-1}\ d$, this should be done as efficiently as possible.

- A classical approach could start by computing the inverse of ξ as the means for calculating the vector e and the matrix M. Then, after applying a threshold process, the linear system $Ma = e$ can be solved.
- The computational cost of the classical approach implies $2N^3 + 2N^2 + 6Nn$ Flops.
- The cost of generating the matrices ϕ and ξ must be added to the previous one.
- However, from the numerical point of view, it should be desirable to obtain M and e without calculating the inverse of ξ.

Ideas to obtain an efficient algorithm:

- Avoid unstable operations like computing inverses or multiplying large matrices. Instead use orthogonal transformations if possible.
- Try to solve large scale problems, having in mind this: use moderately the memory, avoiding unnecessary storage of data, get a moderate execution time.
- Organize the algorithms in such a way that high performance sequential or parallel libraries can be used.

- As $\xi \in R^{N \times N}$ is a symmetric positive definite matrix, Cholesky decomposition can be used to obtain a lower triangular matrix L such that: $\xi = LL^t$.
- Hence, vector e can be expressed as:

$$e = \phi^t \xi^{-1} d = \phi^t L^{-t} L^{-1} d = \phi^t L^{-t} c_1 = \phi^t c_2 \qquad (20)$$

  with $c_1 = L^{-1} d$ and $c_2 = L^{-t} c_1$.
- Thus, vector e can be computed by performing a matrix-vector product, where matrix $\phi \in R^{N \times N}$.
- Observe that these operations involve a cost of $(N^3)/6 + 4(N^2)$ Flops.

- As explained, thresholding can be applied now to vector e, obtaining those positions with a value higher than 4σ.
- This is equivalent to obtain a selection matrix $P \in R^{N\times n}$, which consists of those columns of the identity matrix with a '1' in the position determined by the thresholding of vector e, and obtain $\tilde{e} = P^t e = (\phi P)^t c_2$.
- In order to construct the part of matrix M which is involved in the threshold linear system, we construct

$$\tilde{M} = P^t M P = (\phi P)^t \xi^{-1} (\phi P) = \tilde{\phi}^t L^{-t} L^{-1} \tilde{\phi}, \qquad (21)$$

  with $\tilde{\phi} = \phi P \in R^{N\times n}$.

- Thus,

$$\tilde{M} = (L^{-1}\tilde{\phi})^t (L^{-1}\tilde{\phi}) = Z^t Z, \texttt{ with } Z = L^{-1}\tilde{\phi}.$$

- If we compute the QR decomposition of $Z = QR$, with $Q \in R^{N\times N}$, orthogonal, and $R \in R^{N\times n}$, upper triangular,

$$\tilde{M} = Z^t Z = (QR)^t (QR) = R^t R \Rightarrow Ma = e \Rightarrow (R^t R)a = \tilde{\phi}^t c_2.$$

- The vector a can be computed by solving the triangular linear systems

$$R^t y = \tilde{\phi}^t c_2,\ Ra = y.$$

- The construction of $\tilde{M}$ involves $(N^2)n + 2n^2(N - n/3)$ Flops and the solution of the final linear systems involves $2n^2$ Flops.

Algorithm 5 Algorithm CMB

Input: N, PIX, $FWHM$, $d \in R^{N\times 1}$
Output: a
Step 1. Generate matrices ϕ and ξ
Step 2. Compute $\xi = LL^t$ (Cholesky factorization)
Step 3. Obtain e:
Solve $L * c_1 = d$ and $L^t c_2 = c_1$
Compute $e = \phi^t c_2$
Step 4. Calculate the positions of e that are above the threshold ($e(i) \geq 4\sigma$, $e \rightarrow \tilde{e}$)
Step 5. Get the columns of ϕ associated with the indices from Step 3: $\phi \rightarrow \tilde{\phi}$
Step 6. Solve $LZ = \tilde{\phi}$
Step 7. Compute the QR factorization of Z ($Z = QR$)
Step 8. Solve the triangular systems: $R^t y = \tilde{e}$, $Ra = y$

1.2.3 Implementation details and experimental results

- We call Classical Algorithm to the one that constructs the matrix M and the vector e starting from the inverse of ξ. In turn, the CMB Algorithm avoids the inverse computation and uses matrix decomposition techniques.
- In both algorithms, *OpenMP* is used when possible.
- *OpenMP* is an API that supports multi-platform shared memory multiprocessing programming techniques on most processor architectures and operating systems.
- Arithmetic intensive operations of the algorithms have been addressed through calls to the appropriate subroutines.

Algorithm CMB

Step 1. Generate matrices ϕ and ξ
Step 2. Compute $\xi = LL^t$ (Cholesky factorization) LAPACK
Step 3. Obtain e:
  Solve $L * c_1 = d$ and $L^t c_2 = c_1$ BLAS 2
  Compute $e = \phi^t c_2$ BLAS 2
Step 4. Calculate the positions of e that are above the threshold
Step 5. Get the columns of ϕ associated with the indices from Step 3
Step 6. Solve $LZ = \tilde{\phi}$ BLAS 3
Step 7. Compute the QR factorization of Z ($Z = QR$) LAPACK
Step 8. Solve the triangular systems: $R^t y = \tilde{e}$, $Ra = y$ BLAS 2

- The testbed system used in the experimentation is composed by one Intel Xeon E5530 Quad-Core processors (4 cores) running at 2.40 GHz with Ubuntu Linux distro (10.04.2 LTS) as operating system.
- The high performance implementation of LAPACK was provided by Intel MKL (*Mathematical Kernel Library*, version 10.3).
- Finally, experiments reported employ IEEE 754 double precision arithmetic.
- Coming back to the algorithm described, we can observe an initial step which is the same as for the Classical: Step 1. Generate matrices ϕ and ξ.

- However, the matrices ϕ and ξ that appear in the problem are symmetric Toeplitz-block Toeplitz matrices with symmetric blocks.
- A block matriz A whose (i,j)th block A_{ij} is a function of $(i-j)$ is called block Topelitz matrix. When A_{ij} is itself a Toeplitz matrix, A is called Toeplitz-block Toeplitz matrix.
- Using the features provided for this structure we have obtained an efficient algorithm that greatly reduces the computational cost of the algorithm describes.

- Figure 4 shows the execution times obtained for the algorithms considered.

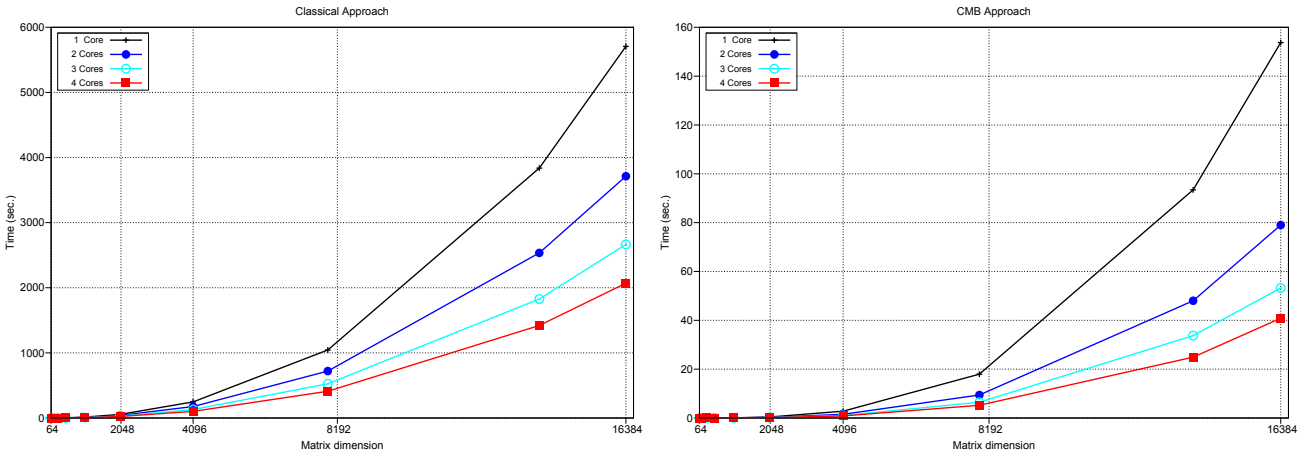

Figura 4: Execution time for the Classical and CMB algorithm.

- The time of CMB algorithm is small. This suggests the possibility of studying wider regions of space which involves the processing of larger matrices at an affordable execution time by using a larger number of cores.

Algorithm/Cores	1	2	3	4
Classical	$5.71e+03$	$3.71e+03$	$2.67e+03$	$2.07e+03$
CMB	$1.54e+02$	$7.90e+01$	$5.32e+01$	$4.09e+01$

Tabla 2: Time (sec.) for $N = 2^{14}$.

1.2.4 Simulations

- In order to check the performance of the new method, we have carried out several simulations resembling the observational data collected by the LFI of the Planck satellite.
- We have simulated data with the characteristics of the 30 GHz channel of the Planck satellite. Our simulations are flat patches of $N = 128 \times 128$ pixels, so that the size of each patch is 14.66×14.66 degrees.
- Each simulated patch consists of several components: a map of point sources, a CMB map and the instrumental noise.

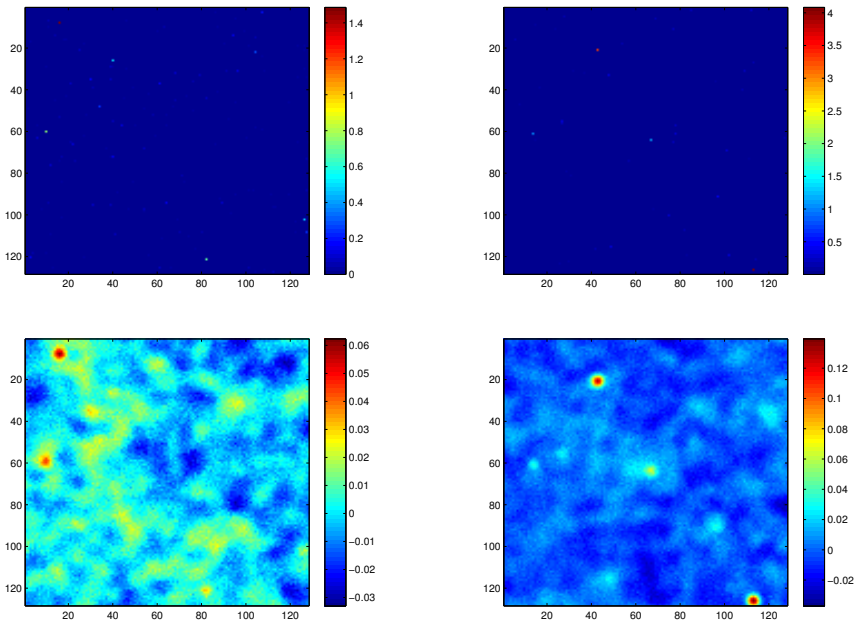

Figura 5: Two simulations at 30 GHz. In the upper panel simulations of point sources and in the lower the same simulations including the CMB plus noise.

Remarks:

- It allows to confront large scale problems with a reasonable execution time, optimizing the memory usage.
- Its parallelization is very efficient; near-optimal speedups have been obtained in many cases.
- The constructed algorithm is scalable in the sense that execution time can be maintained, by increasing the problem size and the number of cores at the same rate.
- The use of high-performance libraries and the organization of the algorithm guarantees numerical stability and portability.
- We have used our new method to detect point sources in simulated CMB maps. These maps resemble the real ones surveyed by the LFI (Planck satellite). The new technique allows us to find the simulated point sources in their positions and to estimate their corresponding fluxes.