

A pipelined parallel OSIC algorithm based on the square root Kalman Filter for heterogeneous networks

F.J. Martínez-Zaldívar¹, A.M. Vidal-Maciá², D. Giménez³

¹*Dept. Comunicaciones, Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia (Spain)
fjmartin@dcom.upv.es*

²*Dept. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia (Spain)
avidal@dsic.upv.es*

³*Departamento de Informática y Sistemas, Universidad de Murcia
Campus de Espinardo, 30071 Murcia (Spain)
domingo@dif.um.es*

Abstract—This paper describes a pipelined parallel algorithm for the Ordered Successive Interference Cancellation (OSIC) decoding procedure proposed in V-BLAST wireless MIMO systems. It is based on an algorithm that solves the Recursive Least Squares (RLS) problem, and is derived from a block version of the square root version of the Kalman Filter. It has been parallelized in a pipelined way getting a good efficiency and scalability in a heterogeneous network of computers. Although the optimum load balancing for this algorithm is dynamic, we derive a static load balancing scheme with good results.

Index Terms—OSIC, MIMO, V-BLAST, pipeline, Kalman

I. INTRODUCTION

Multiple Input Multiple Output (MIMO) systems have been extensively studied in the context of wireless communications in the recent years. The original proposal by Foschini, [1], known as BLAST (Bell Labs Layered Space-Time), has generated a family of architectures that uses multiple antenna arrays to transmit and receive information with the target of increasing the capacity and reliability of the links. Some of these architectures are D-BLAST (Diagonal BLAST), [2], [1] and Turbo-BLAST, [3], with high complexity implementations, and V-BLAST (Vertical BLAST) with more practical complexity at expense of diversity, [4]. We focus our interest in the suboptimal but more practical V-BLAST family where nearly optimal decoders as Sphere Decoding can be used, [5], or linear decoders as Zero Forcing, MMSE and its ordered version OSIC (Ordered Successive Interference Cancellation), [6], [7], with some applications like multicarrier systems (OFDM used in DVB-T —Digital Video Broadcasting-Terrestrial—), [8] where the dimension of the problem may be about several thousands.

This paper describes the parallelization of the algorithm that solves the OSIC decoding problem for a heterogeneous

networks of computers. The parallel algorithm can be used with the basic ideas of [6] and the improvement of [7].

II. OSIC DECODING PROCEDURE

In a basic approach, we need to solve the typical perturbed system $\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{v}$, where the known full rank matrix $\mathbf{H} \in \mathbb{C}^{m \times n}$ represents the channel matrix and any manipulation of the symbols before transmission, \mathbf{x} is a vector whose components belong to a discrete symbol set, and \mathbf{v} is the process noise. The use of MMSE (Minimum Mean Square Error estimation) yields

$$\hat{\mathbf{x}} = \left[\left(\begin{array}{c} \mathbf{H} \\ \sqrt{\alpha}\mathbf{I}_n \end{array} \right)^\dagger \left(\begin{array}{c} \mathbf{y} \\ \mathbf{0} \end{array} \right) \right] = [\mathbf{H}_\alpha^\dagger \mathbf{y}] \quad (1)$$

where $[\cdot]$ denotes the mapping or slicing of the result in the symbol set, $\mathbf{H}_\alpha^\dagger$ denotes the first m columns of the pseudoinverse of the augmented channel matrix $(\mathbf{H}^*, \sqrt{\alpha}\mathbf{I}_n)^*$, α^{-1} may be thought as a signal to noise ratio, and the asterisk superscript $(\cdot)^*$ denotes the complex conjugate. In OSIC, the signal components x_i , $i = 1, \dots, n$, are decoded from the strongest (with highest signal to noise ratio) to the weakest, cancelling the contribution of the decoded signal component to the received signal and then repeating the process with the remainder signal components. Let \mathbf{P} be the error estimation covariance matrix (symmetric positive definite)

$$\mathbf{P} = \mathbb{E}\{(\mathbf{x} - \hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}})^*\} = (\alpha\mathbf{I}_n + \mathbf{H}^*\mathbf{H})^{-1}$$

We can verify that

$$\mathbf{P} = \left(\begin{array}{c} \mathbf{H} \\ \sqrt{\alpha}\mathbf{I}_n \end{array} \right)^\dagger \left(\left(\begin{array}{c} \mathbf{H} \\ \sqrt{\alpha}\mathbf{I}_n \end{array} \right)^\dagger \right)^* \quad (2)$$

Later, we will need to factorize \mathbf{P} as the product of their square roots factors. This factorization is not unique but it is advantageous that these factors have triangular structure. Let $\mathbf{P} = \mathbf{P}^{1/2}\mathbf{P}^{*/2}$ be the Cholesky factorization of \mathbf{P} , then the square root factor $\mathbf{P}^{1/2}$ is a lower triangular matrix. Let \mathbf{P}_{jj} be $\mathbf{P}_{jj} = \min\{\text{diag}(\mathbf{P})\}$ (so the j^{th} row of $\mathbf{P}^{1/2}$ has the minimum euclidean norm) then \hat{x}_j is the component of $\hat{\mathbf{x}}$ with the highest signal to noise ratio that can be decoded as $\hat{x}_j = \left[\mathbf{H}_{\alpha,j}^\dagger \mathbf{y} \right]$, where $\mathbf{H}_{\alpha,j}^\dagger$ is the j^{th} row of $\mathbf{H}_\alpha^\dagger$ (the j^{th} nulling vector). If $\mathbf{H} = (\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_n)$, the contribution of the estimated x_j in the received signal is cancelled as $\mathbf{y}' = \mathbf{y} - \mathbf{h}_j \hat{x}_j$.

The procedure should be repeated for the deflated channel matrix \mathbf{H}' , which is obtained by deleting the j^{th} column of \mathbf{H} . It implies that we should recompute the pseudoinverse of the *augmented* deflated channel matrix and then select the new strongest component signal, applying the same calculations. The computation of the iteratively augmented deflated channel matrix pseudoinverses until all the signal components are decoded, represents a high computational cost. This recomputation is avoided in [6]. Let us compute the next QL factorization:

$$\left(\begin{array}{c} \mathbf{H} \\ \sqrt{\alpha} \mathbf{I}_n \end{array} \right) = \mathbf{QL} \Rightarrow \left(\begin{array}{c} \mathbf{H} \\ \sqrt{\alpha} \mathbf{I}_n \end{array} \right)^\dagger = \mathbf{L}^{-1} \mathbf{Q}^* \quad (3)$$

where $\mathbf{L} \in \mathbb{C}^{n \times n}$ is a lower triangular matrix, and the columns of $\mathbf{Q} \in \mathbb{C}^{(m+n) \times n}$ are orthogonal. Let us define $\mathbf{Q} = (\mathbf{Q}_\alpha^*, \mathbf{Q}_\beta^*)^*$, where $\mathbf{Q}_\alpha = (\mathbf{q}_{\alpha,1}, \mathbf{q}_{\alpha,2}, \dots, \mathbf{q}_{\alpha,n})$ are the first m rows of \mathbf{Q} , then we can identify from (2) and (3) that $\mathbf{L} = \mathbf{P}^{-1/2}$. Hence:

$$\begin{aligned} \mathbf{P}^{1/2} &= \begin{pmatrix} p_1^{1/2} & 0 & \dots & 0 \\ \times & p_2^{1/2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \times & \times & \dots & p_n^{1/2} \end{pmatrix} \\ \mathbf{P}^{-1/2} &= \mathbf{L} \begin{pmatrix} p_1^{-1/2} & 0 & \dots & 0 \\ * & p_2^{-1/2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \dots & p_n^{-1/2} \end{pmatrix} \\ &= \begin{pmatrix} p_1^{-1/2} & \mathbf{0} \\ \mathbf{a} & \mathbf{L}' \end{pmatrix} \end{aligned} \quad (4)$$

verifying that

$$\mathbf{H}_\alpha^\dagger = \mathbf{P}^{1/2} \mathbf{Q}_\alpha^* \quad (5)$$

Let us suppose that the first row of the lower triangular matrix $\mathbf{P}^{1/2}$ is the least euclidean norm row (otherwise we can get it using a permutation matrix $\mathbf{\Pi}$ and a unitary transformation $\mathbf{\Sigma}$ in $\mathbf{\Pi} \mathbf{H}_\alpha^\dagger = \mathbf{\Pi} \mathbf{P}^{1/2} \mathbf{\Sigma} \mathbf{\Sigma}^* \mathbf{Q}_\alpha^*$ maintaining the lower triangular structure in $\mathbf{\Pi} \mathbf{P}^{1/2} \mathbf{\Sigma}$); then the first

diagonal entry of $\mathbf{P} = \mathbf{P}^{1/2} \mathbf{P}^{*/2}$ would be the smallest one. Hence the first component to decode (the strongest one) would be \hat{x}_1 , so the nulling vector $\mathbf{H}_{\alpha,1}^\dagger$ can be got from (5) as $\mathbf{H}_{\alpha,1}^\dagger = p_1^{1/2} \mathbf{Q}_{\alpha,1}^*$ due to the lower triangular structure of $\mathbf{P}^{1/2}$, so $\hat{x}_1 = \left[\mathbf{H}_{\alpha,1}^\dagger \mathbf{y}^{(n)} \right]$, where $\mathbf{y}^{(n)} = \mathbf{y}$. Now we must cancel the contribution of \hat{x}_1 to the received signal as $\mathbf{y}^{(n-1)} = \mathbf{y}^{(n)} - \mathbf{h}_1 \hat{x}_1$ and repeat the process with the new strongest signal component, the cancelled interference received signal $\mathbf{y}^{(n-1)}$ and the deflated channel matrix $\mathbf{H}' = (\mathbf{h}_2, \dots, \mathbf{h}_n)$, and so on. The deflated channel matrix \mathbf{H}' has a new error covariance matrix $\mathbf{P}' = \mathbf{P}^{1/2'} \mathbf{P}'^{*/2'}$ that can be obtained directly from the last $n-1$ columns and rows of $\mathbf{P}^{1/2}$. From (3) and (4):

$$\begin{aligned} \left(\begin{array}{c} \mathbf{H} \\ \sqrt{\alpha} \mathbf{I}_n \end{array} \right) &= \left(\begin{array}{cccc} \mathbf{h}_1 & \mathbf{h}_2 & \dots & \mathbf{h}_n \\ & & & \sqrt{\alpha} \mathbf{I}_n \end{array} \right) \\ &= \mathbf{QL} \\ &= \begin{pmatrix} \mathbf{Q}_\alpha \\ \mathbf{Q}_\beta \end{pmatrix} \begin{pmatrix} p_1^{-1/2} & \mathbf{0} \\ \mathbf{a} & \mathbf{L}' \end{pmatrix} \end{aligned}$$

we get

$$\begin{aligned} (\mathbf{h}_1 \quad \mathbf{h}_2 \quad \dots \quad \mathbf{h}_n) &= \mathbf{Q}_\alpha \begin{pmatrix} p_1^{-1/2} & \mathbf{0} \\ \mathbf{a} & \mathbf{L}' \end{pmatrix} \\ &= (\mathbf{q}_{\alpha,1}, \mathbf{q}_{\alpha,2}, \dots, \mathbf{q}_{\alpha,n}) \cdot \begin{pmatrix} p_1^{-1/2} & \mathbf{0} \\ \mathbf{a} & \mathbf{L}' \end{pmatrix} \\ (\mathbf{h}_2 \quad \dots \quad \mathbf{h}_n) &= (\mathbf{q}_{\alpha,2}, \dots, \mathbf{q}_{\alpha,n}) \mathbf{L}' \\ \left(\begin{array}{c} \mathbf{H}' \\ \sqrt{\alpha} \mathbf{I}_{n-1} \end{array} \right) &= \begin{pmatrix} \mathbf{Q}'_\alpha \\ \mathbf{Q}'_\beta \end{pmatrix} \mathbf{L}' \end{aligned}$$

Then we can get the new strongest component signal from this new $\mathbf{P}'^{1/2'} = \mathbf{L}'^{-1}$ matrix. The new \mathbf{Q}'_α and \mathbf{Q}'_β can be obtained directly from the last $n-1$ columns of \mathbf{Q}_α and \mathbf{Q}_β respectively.

Therefore, only $\mathbf{P}^{1/2}$ and \mathbf{Q}_α are necessary to solve the OSIC problem. In [6], $\mathbf{P}^{1/2}$ and \mathbf{Q}_α are computed solving $\hat{\mathbf{x}} = \left[\mathbf{H}_\alpha^\dagger \mathbf{y} \right] = \left[\mathbf{P}^{1/2} \mathbf{Q}_\alpha^* \mathbf{y} \right]$ as a Recursive Least Squares (RLS) problem in a special way. A subtle improvement in the execution time is reported in [7] where the nulling vector is got by means $\mathbf{P}^{1/2}$ and \mathbf{H} . Anyway, in both cases we need the $\mathbf{P}^{1/2}$ matrix. We will develop the parallel algorithm for the ideas reported in [6] because the results are extrapolable to the implementation proposed in [7]. In order to simplify the description of the algorithm we will avoid the details of the permutations due to the signal to noise ratio orderings and will focus our attention on the parallelization of the process for obtaining $\mathbf{P}^{1/2}$ and \mathbf{Q}_α .

A. The square root Kalman Filter for OSIC

From (5), $\mathbf{Q}_\alpha = \mathbf{H}_\alpha^\dagger \mathbf{P}^{-*/2}$. This matrix is propagated along the iterations of the square root Kalman Filter devised initially to solve a Recursive Least Squares (RLS) problem. Next we reproduce a block version of the algorithm for OSIC, reported in [6] that will be named SRKF-OSIC.

Input: $\mathbf{H} = (\mathbf{H}_0^*, \mathbf{H}_1^*, \dots)^*$, $\mathbf{P}_{(0)}^{1/2} = \frac{1}{\sqrt{\alpha}} \mathbf{I}_n$ and $\mathbf{Q}_{\alpha,(0)} = \mathbf{0}$

Output: $\mathbf{Q}_\alpha = \mathbf{Q}_{\alpha,(m/q)}$, $\mathbf{P}^{1/2} = \mathbf{P}_{(m/q)}^{1/2}$

for $i = 0, \dots, m/q - 1$ do

Calculate $\Theta_{(i)}$ and applicate in such a way that:

$$\begin{aligned} \mathbf{E}_{(i)} \Theta_{(i)} &= \begin{pmatrix} \mathbf{I}_q & \mathbf{H}_i \mathbf{P}_{(i)}^{1/2} \\ \mathbf{0} & \mathbf{P}_{(i)}^{1/2} \\ -\Gamma_{(i+1)} & \mathbf{Q}_{\alpha,(i)} \end{pmatrix} \Theta_{(i)} \\ &= \begin{pmatrix} \mathbf{R}_{e,(i)}^{1/2} & \mathbf{0} \\ \bar{\mathbf{K}}_{p,(i)} & \mathbf{P}_{(i+1)}^{1/2} \\ \mathbf{Z} & \mathbf{Q}_{\alpha,(i+1)} \end{pmatrix} = \mathbf{F}_{(i)} \end{aligned}$$

end for

where $\mathbf{Z} = -\left(\Gamma_{(i+1)}^* - \mathbf{H}_i \mathbf{H}_{\alpha,(i+1)}^\dagger\right)^* \mathbf{R}_{e,i}^{-*/2}$, q is the number of consecutive rows of \mathbf{H} processed in a block, so $\mathbf{H}_i \in \mathbb{C}^{q \times n}$ is the i^{th} block of consecutive rows of \mathbf{H} ; the iteration index subscript enclosed between parenthesis denotes that the variable is updated iteratively: \mathbf{Q}_α and $\mathbf{P}^{1/2}$ are the values $\mathbf{Q}_{\alpha,(i+1)}$ and $\mathbf{P}_{(i+1)}^{1/2}$ in the last iteration $i = m/q - 1$; $\Gamma_{(i+1)} = \left(\mathbf{0}_{iq \times q}^T, \mathbf{I}_q, \mathbf{0}_{(m-q(i+1)) \times q}^T\right)^T \in \mathbb{R}^{m \times q}$; $\mathbf{R}_{e,i}$ and $\bar{\mathbf{K}}_{p,(i)}$ are variables of the Kalman Filter whose meaning are described in [9], and $\mathbf{H}_{\alpha,(i+1)}^\dagger$ appears implicitly in \mathbf{Z} (this submatrix will not be propagated along the iterations as can be observed from the algorithm).

We can use a QR factorization in order to get the required zeroes but we can get a lower computational cost if we propagate the matrix $\mathbf{P}_{(i)}^{1/2}$ maintaining a lower triangular structure along the iterations, so the zeros must be get in a selective way using Givens rotations. We can exploit the fact that the structure of Γ_{i+1} depends on the iteration index i and $\mathbf{Q}_{\alpha,(0)} = \mathbf{0}$, in order to minimize the arithmetic cost of the unitary matrix application.

1) *Arithmetic costs:* The costs of one iteration are a matrix multiplication $\mathbf{H}_i \mathbf{P}_{(i)}^{1/2}$ and the application of a sequence of Givens rotations $\Theta_{(i)}$, exploiting and maintaining the triangular structure of $\mathbf{P}_{(i)}^{1/2}$ along the iterations. Let $w_{\text{TRMM}}(q, n)$ denote the cost of the $\mathbf{H}_i \mathbf{P}_{(i)}^{1/2}$ matrix multiplication, where q and n are the dimensions of the result, and $w_{\text{ROT}}(z)$, the cost of applying a Givens rotation to a pair of vectors of z components. The cost can be approximated as:

$$\begin{aligned} W_{\text{seq},i}(n, q) &= w_{\text{TRMM}}(q, n) + \sum_{c=1}^n \sum_{r=1}^q [w_{\text{ROT}}(q-r+1) \\ &\quad + w_{\text{ROT}}(n-c+1) + w_{\text{ROT}}([i+1]q)] \\ &= w_{\text{TRMM}}(q, n) + \left[n \sum_{r=1}^q w_{\text{ROT}}(r) \right. \\ &\quad \left. + q \sum_{c=1}^n w_{\text{ROT}}(c) + qn w_{\text{ROT}}([i+1]q) \right] \end{aligned}$$

where r , c and i denotes the index of rows, columns and iterations respectively. If we assume that $w_{\text{TRMM}}(q, n) = qn^2$ flops and $w_{\text{ROT}}(z) = 6z$ flops, [10], the cost of the i^{th} iteration is

$$\begin{aligned} W_{\text{seq},i}(n, q) &= qn^2 + [3(n+q+2+2q[i+1])qn] \\ &= (4n+3q+6+6q[i+1])qn \quad \text{flops (6)} \end{aligned}$$

and the cost of the total iterations is

$$W_{\text{seq}}(m, n, q) = \sum_{i=0}^{m/q-1} W_{\text{seq},i}(n, q) \approx 4n^2m + 3nm^2 \quad \text{flops}$$

III. PARALLEL ALGORITHM

A. Data decomposition

For clarity reasons let us suppose as an example that we have $p = 3$ processors: P_0 , P_1 and P_2 . A matrix enclosed within square brackets with a processor subscript will denote that part of the matrix belongs to such processor. If it is enclosed within parenthesis then it denotes that the entire matrix is in such processor.

Let $(\mathbf{D}_{(i)} | \mathbf{C}_{(i)})$ denote the generic structure of $\mathbf{E}_{(i)}$ and $\mathbf{F}_{(i)}$

$$\begin{aligned} \mathbf{E}_{(i)} &= \left(\begin{array}{c|c} \mathbf{I}_q & \mathbf{H}_i \mathbf{P}_{(i)}^{1/2} \\ \mathbf{0} & \mathbf{P}_{(i)}^{1/2} \\ -\Gamma_{(i+1)} & \mathbf{Q}_{\alpha,(i)} \end{array} \right) \Rightarrow (\mathbf{D}_{(i)} | \mathbf{C}_{(i)}) \\ \mathbf{F}_{(i)} &= \left(\begin{array}{c|c} \mathbf{R}_{e,(i)}^{1/2} & \mathbf{0} \\ \bar{\mathbf{K}}_{p,(i)} & \mathbf{P}_{(i+1)}^{1/2} \\ \mathbf{Z} & \mathbf{Q}_{\alpha,(i+1)} \end{array} \right) \Rightarrow (\mathbf{D}_{(i)} | \mathbf{C}_{(i)}) \end{aligned}$$

We will observe later that $\mathbf{D}_{(i)} \in \mathbb{C}^{(q+n+m) \times q}$ will be manipulated by all the processors in a pipelined way so we will denote it as $(\mathbf{D}_{(i)})_{P_j}$, where P_j represents the processors that processes it. $\mathbf{D}_{(i)}$ is made up by a lower triangular matrix $\mathbf{L}_{(i)} \in \mathbb{C}^{q \times q}$ and two general matrices $\mathbf{M}_{(i)} \in \mathbb{C}^{n \times q}$ and $\mathbf{N}_{(i)} \in \mathbb{C}^{m \times q}$. $\mathbf{M}_{(i)} \in \mathbb{C}^{n \times q}$ will be divided by rows in p groups denoting a left superscript the amount of rows of the grouping:

$$(\mathbf{D}_{(i)})_{P_j} = \begin{pmatrix} \mathbf{L}_{(i)} \\ \mathbf{M}_{(i)} \\ \mathbf{N}_{(i)} \end{pmatrix} = \begin{pmatrix} \mathbf{L}_{(i)} \\ n_2 [\mathbf{M}_{(i)}] \\ n_1 [\mathbf{M}_{(i)}] \\ n_0 [\mathbf{M}_{(i)}] \\ \mathbf{N}_{(i)} \end{pmatrix}$$

with $\sum_{j=0}^{p-1} n_j = n$. The nonzero rows of $\mathbf{N}_{(i)}$ will depend on the iteration index i . Initially:

$$(\mathbf{D}_{(i)})_{P_0} = \begin{pmatrix} \mathbf{L}_{(i)} \\ \mathbf{M}_{(i)} \\ \mathbf{N}_{(i)} \end{pmatrix} = \begin{pmatrix} \mathbf{I}_q \\ \mathbf{0} \\ -\Gamma_{i+1} \end{pmatrix}$$

And at the end of the process and as consequence of the unitary matrix application:

$$(\mathbf{D}^{(i)})_{P_{p-1}} = \begin{pmatrix} \mathbf{L}^{(i)} \\ \mathbf{M}^{(i)} \\ \mathbf{N}^{(i)} \end{pmatrix} = \begin{pmatrix} \mathbf{R}_{e,(i)}^{1/2} \\ \overline{\mathbf{K}}_{p,(i)} \\ \mathbf{Z} \end{pmatrix}$$

The last n columns of $\mathbf{E}^{(i)}$ and $\mathbf{F}^{(i)}$ will be the initial and final value respectively of $\mathbf{C}^{(i)}$. This submatrix will be distributed by columns among the processors. Every one will own n_j consecutive columns with $\sum_{j=0}^{p-1} n_j = n$. c_{0_j} will denote the index of the first of them assigned to the P_j processor. We can verify that

$$c_{0_j} = 1 + \sum_{k=j+1}^{p-1} n_k = n + 1 - \sum_{k=j}^{p-1} n_k \quad (7)$$

This data distribution of $\mathbf{C}^{(i)}$ will not change in the algorithm execution (except with an adaptive load balancing).

Then

$$\begin{aligned} \mathbf{E}^{(i)} &= \begin{pmatrix} \mathbf{D}^{(i)} & \mathbf{C}^{(i)} \end{pmatrix} \\ &= \begin{pmatrix} (\mathbf{D}^{(i)})_{P_0} & [\mathbf{C}^{(i)}]_{P_2} & [\mathbf{C}^{(i)}]_{P_1} & [\mathbf{C}^{(i)}]_{P_0} \end{pmatrix} \end{aligned}$$

B. Processors tasks

When the P_j processor gets zeroes in $[\mathbf{H}\mathbf{P}_i^{1/2}]_{P_j}$ for the i^{th} iteration, we can observe that $[\mathbf{P}_{(i)}^{1/2}]_{P_j}$ and $[\mathbf{Q}_{\alpha,(i)}]_{P_j}$ are converted in $[\mathbf{P}_{(i+1)}^{1/2}]_{P_j}$ and $[\mathbf{Q}_{\alpha,(i+1)}]_{P_j}$ respectively, so if P_j has got \mathbf{H}_{i+1} , it can get $[\mathbf{C}_{i+1}]_{P_j}$ and begin with the process for the $(i+1)^{\text{th}}$ iteration. The pipelined behaviour of the algorithm is based on this fact.

The first step in the pipelined parallel algorithm is (an apostrophe will denote the updating of the variable):

$$\begin{aligned} \mathbf{E}'^{(i)} &= \mathbf{E}^{(i)} \Theta_{i,P_0} \\ &= \begin{pmatrix} (\mathbf{D}^{(i)})_{P_0} & [\mathbf{C}^{(i)}]_{P_2} & [\mathbf{C}^{(i)}]_{P_1} & [\mathbf{C}^{(i)}]_{P_0} \end{pmatrix} \cdot \Theta_{i,P_0} \\ &= \left(\begin{pmatrix} \mathbf{I}_q \\ n_2[\mathbf{0}] \\ n_1[\mathbf{0}] \\ n_0[\mathbf{0}] \\ -\mathbf{F}^{(i+1)} \end{pmatrix}_{P_0} \begin{pmatrix} [\mathbf{C}^{(i)}]_{P_2} & [\mathbf{C}^{(i)}]_{P_1} \\ [\mathbf{C}^{(i)}]_{P_0} \end{pmatrix} \Theta_{i,P_0} \right) \\ &= \left(\begin{pmatrix} \mathbf{L}^{(i)} \\ n_2[\mathbf{0}] \\ n_1[\mathbf{0}] \\ n_0[\mathbf{M}^{(i)}] \\ \mathbf{N}^{(i)} \end{pmatrix}_{P_0} \begin{pmatrix} [\mathbf{C}^{(i)}]_{P_2} & [\mathbf{C}^{(i)}]_{P_1} \\ [\mathbf{C}^{(i)}]_{P_0} \end{pmatrix} \Theta_{i,P_0} \right) \\ &= \left(\begin{pmatrix} \mathbf{L}^{(i)} \\ n_2[\mathbf{0}] \\ n_1[\mathbf{0}] \\ n_0[\mathbf{M}^{(i)}] \\ \mathbf{N}^{(i)} \end{pmatrix}_{P_0} \begin{pmatrix} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \\ \mathbf{Q}_{\alpha,(i+1)} \end{pmatrix}_{P_0} \right) \end{aligned}$$

where Θ_{i,P_0} is a unitary transformation calculated and applied by P_0 in order to get zeros in $[\mathbf{H}_i \mathbf{P}_i^{1/2}]_{P_0}$. Now P_0 must transfer the nonzero part of $(\mathbf{D}^{(i)})_{P_0}$ to P_1 : $\mathbf{L}^{(i)}$ (a lower triangular matrix, $q(q+1)/2$ elements), $n_0[\mathbf{M}^{(i)}]$ ($n_0 q$ elements) and the nonzero part of $\mathbf{N}^{(i)}$ ($(i+1)q$ elements). If P_0 has received \mathbf{H}_{i+1} , it has all necessary data to make up $[\mathbf{C}_{i+1}]_{P_0}$ and start again.

Now in P_1 :

$$\begin{aligned} \mathbf{E}''^{(i)} &= \mathbf{E}'^{(i)} \Theta_{i,P_1} \\ &= \left(\begin{pmatrix} \mathbf{L}^{(i)} \\ n_2[\mathbf{0}] \\ n_1[\mathbf{0}] \\ n_0[\mathbf{M}^{(i)}] \\ \mathbf{N}^{(i)} \end{pmatrix}_{P_1} \begin{pmatrix} [\mathbf{C}^{(i)}]_{P_2} \\ [\mathbf{C}^{(i)}]_{P_1} \begin{bmatrix} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \\ \mathbf{Q}_{\alpha,(i+1)} \end{bmatrix}_{P_0} \end{pmatrix} \Theta_{i,P_1} \right) \\ &= \left(\begin{pmatrix} \mathbf{L}'^{(i)} \\ n_2[\mathbf{0}] \\ n_1[\mathbf{M}^{(i)}] \\ n_0[\mathbf{M}^{(i)}]' \\ \mathbf{N}'^{(i)} \end{pmatrix}_{P_1} \begin{pmatrix} [\mathbf{C}^{(i)}]_{P_2} \\ [\mathbf{C}^{(i)}]_{P_1} \begin{bmatrix} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \\ \mathbf{Q}_{\alpha,(i+1)} \end{bmatrix}_{P_1} \begin{bmatrix} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \\ \mathbf{Q}_{\alpha,(i+1)} \end{bmatrix}_{P_0} \end{pmatrix} \Theta_{i,P_1} \right) \end{aligned}$$

The same comments are applicable to P_1 . Now in P_2 :

$$\begin{aligned} \mathbf{E}'''^{(i)} &= \mathbf{E}''^{(i)} \Theta_{i,P_2} \\ &= \left(\begin{pmatrix} \mathbf{L}'^{(i)} \\ n_2[\mathbf{0}] \\ n_1[\mathbf{M}^{(i)}] \\ n_0[\mathbf{M}^{(i)}]' \\ \mathbf{N}'^{(i)} \end{pmatrix}_{P_2} \begin{pmatrix} [\mathbf{C}^{(i)}]_{P_2} \\ [\mathbf{C}^{(i)}]_{P_1} \begin{bmatrix} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \\ \mathbf{Q}_{\alpha,(i+1)} \end{bmatrix}_{P_1} \begin{bmatrix} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \\ \mathbf{Q}_{\alpha,(i+1)} \end{bmatrix}_{P_0} \end{pmatrix} \Theta_{i,P_2} \right) \end{aligned}$$

Hence,

$$\mathbf{E}'''_{(i)} = \left(\left(\begin{array}{c} \mathbf{L}''_{(i)} \\ n_2 [\mathbf{M}_{(i)}] \\ n_1 [\mathbf{M}_{(i)}]' \\ n_0 [\mathbf{M}_{(i)}]'' \\ \mathbf{N}''_{(i)} \end{array} \right)_{P_2} \left[\begin{array}{c} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \\ \mathbf{Q}_{\alpha,(i+1)} \end{array} \right]_{P_2} \right. \\ \left. \left(\begin{array}{c} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \\ \mathbf{Q}_{\alpha,(i+1)} \end{array} \right)_{P_1} \left(\begin{array}{c} \mathbf{0} \\ \mathbf{P}_{i+1}^{1/2} \\ \mathbf{Q}_{\alpha,(i+1)} \end{array} \right)_{P_0} \right) \\ = \left(\begin{array}{cc} \mathbf{R}_{e,i}^{1/2} & \mathbf{0} \\ \overline{\mathbf{K}}_{p,i} & \mathbf{P}_{i+1}^{1/2} \\ \mathbf{Z} & \mathbf{Q}_{\alpha,(i+1)} \end{array} \right) = \mathbf{F}_i$$

Figure 1 depicts the behavior of the parallel algorithm.

C. Arithmetic cost

The arithmetic cost in the i^{th} iteration in the P_j processor is due to:

- The matrix multiplication $[\mathbf{H}_i \mathbf{P}_i^{1/2}]_{P_j}$ can be done taking advantage of the $[\mathbf{P}_i^{1/2}]_{P_j}$ matrix structure:

$$\text{WTRMM}(q, n_j) + \text{Wgemm}(q, n - c_{0_j} - n_j + 1, n_j) \\ + \text{WADD}(q, n_j)$$

Let us suppose that the cost of the two general $a \times b$ and $b \times c$ matrix multiplication is $\text{Wgemm}(a, b, c) = a(2b - 1)c$ flops, and the cost of two $a \times b$ matrices addition is $\text{WADD}(a, b) = ab$ flops. Hence, the cost is:

$$qn_j^2 + q(2n - 2c_{0_j} - 2n_j + 1)n_j + qn_j = \\ (2n - n_j - 2c_{0_j} + 2)qn_j \quad \text{flops} \quad (8)$$

- The zeroes in the n_j columns of $[\mathbf{H}_i \mathbf{P}_i^{1/2}]_{P_j}$ are got column-wise and from right to left. Let $r = 1, \dots, q$ denote the row index in the $c = n_j, \dots, 1$ column of $[\mathbf{H}_i \mathbf{P}_i^{1/2}]_{P_j}$. Then the Givens rotation must be applied to three pair of subvectors:
 - The row interval $[r : q]$ of the r^{th} column of $(\mathbf{L})_{P_j}$ and the c^{th} column of $[\mathbf{H}_i \mathbf{P}_i^{1/2}]_{P_j}$.
 - The row interval $[c_{0_j} + c - 1 : n]$ of the r^{th} column of $(\mathbf{M})_{P_j}$ and the c^{th} column of $[\mathbf{P}_i^{1/2}]_{P_j}$.
 - The row interval $[1 : [i + 1]q]$ of the r^{th} column of $(\mathbf{N}_i)_{P_j}$ and the c^{th} column of $[\mathbf{Q}_{\alpha,(i)}]_{P_j}$.
- so the cost of the i^{th} iteration is:

$$\sum_{c=1}^{n_j} \sum_{r=1}^q \text{wROT}(q - r + 1) + \sum_{c=1}^{n_j} \sum_{r=1}^q \text{wROT}(n - c_{0_j} - c + 2) \\ + \sum_{c=1}^{n_j} \sum_{r=1}^q \text{wROT}([i + 1]q) =$$

$$n_j \sum_{r=1}^q \text{wROT}(r) + q \sum_{c=1}^{n_j} \text{wROT}(n - c_{0_j} - c + 2) \\ + qn_j \text{wROT}([i + 1]q)$$

or

$$(3q + 6n - 6c_{0_j} + 12 - 3n_j + 6[i + 1]q)qn_j \quad \text{flops} \quad (9)$$

So the total cost of the i^{th} iteration is (8) plus (9):

$$W_{P_j,i}(n, q) = (3q + 8n - 8c_{0_j} - 4n_j + 14 + 6[i + 1]q)qn_j \quad \text{flops} \quad (10)$$

and the total arithmetic time in the P_j processor is

$$W_{P_j}(m, n, q) = \sum_{i=0}^{m/q-1} W_{P_j,i}(n, q) \\ = \sum_{i=0}^{m/q-1} \{3q + 8n - 8c_{0_j} - 4n_j + 14 \\ + 6[i + 1]q\}qn_j$$

We can verify that the parallelization arithmetic overhead is null:

$$W_{\text{seq}}(m, n, q) = \sum_{j=0}^{p-1} W_{P_j}(m, n, q)$$

D. Load balancing

The perfect load balancing is got when all processors need the same time in processing its assigned iteration job due to the communication synchronization. We need to determine the n_j value for every P_j in order to balance the load taking into account the different computing power in a heterogeneous network.

Each processor is processing a different iteration at the same time: if P_j is processing the i^{th} iteration, P_k is processing the $(i + j - k)^{\text{th}}$ iteration. So we must achieve

$$W_{P_j,i}(n, q)t_{w_j} = W_{P_k,i+j-k}(n, q)t_{w_k} \quad (11)$$

where t_{w_j} and t_{w_k} are the time per flop in P_j and P_k respectively.

It is difficult or impossible to get the n_j and n_k values, $\forall j \neq k$ and $\forall i$, that verify (11) with the constraint $\sum_{j=0}^{p-1} n_j = n$. Hence, we will relax the condition (11) with a simpler target: $W_{P_j,i}(n, q)t_{w_j} = W_{P_k,i}(n, q)t_{w_k}$. The $n_j, \forall j$ values can be got solving the next equation:

$$W_{P_j,i}(n, q)t_{w_j} = \frac{W_{\text{seq},i}(n, q)t_{w_{\text{seq}}}}{S_{\text{max}}(p, n, q)}, \forall 0 \leq j \leq p - 1 \quad (12)$$

where $S_{\text{max}}(p, n, q)$ is the maximum speedup attainable in the parallel system.

The maximum speedup in the heterogeneous network depends on the time per flop t_{w_j} of every processor. Let us

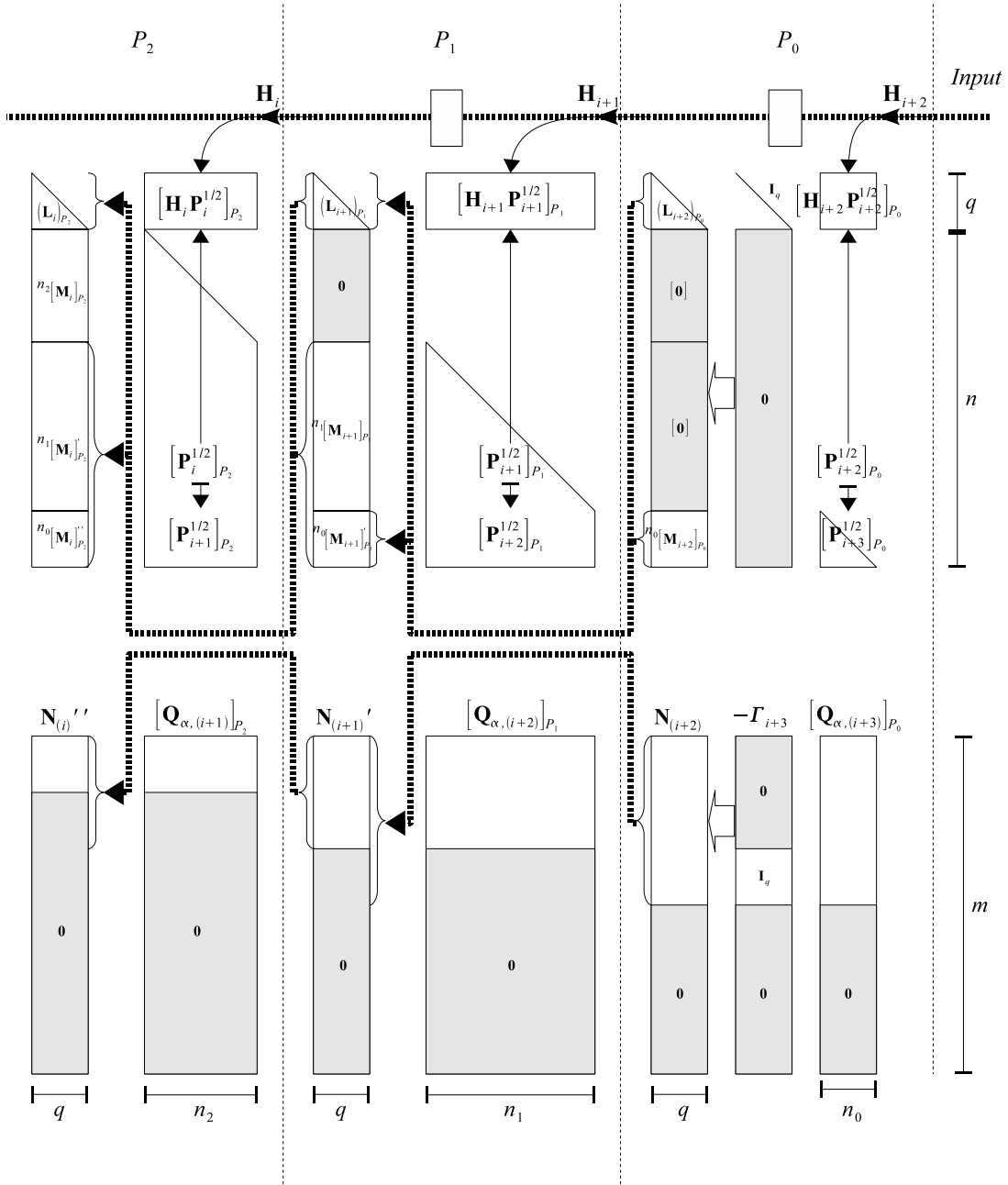


Fig. 1. SRKF-OSIC pipelined parallel algorithm.

define s_j as the normalized relative speed of the processors (dimensionless):

$$s_j = \frac{1}{\sum_{r=0}^{p-1} \frac{t_{w_j}}{t_{w_r}}} \quad (13)$$

We can verify that $\sum_{j=0}^{p-1} s_j = 1$, and $t_{w_j} s_j = t_{w_k} s_k$, and if P_j is u times faster than P_k , then $s_j = u s_k$.

Let us suppose that the sequential algorithm is run in the fastest processor of the heterogeneous network: P_f , $0 \leq f \leq p-1$, then $t_{w_{\text{seq}}} = t_{w_f}$. The maximum speedup can be obtained from (12) when a perfect load

balancing is got ($W_{P_j,i}(n,q)t_{w_j} = W_{P_k,i}(n,q)t_{w_k}, \forall j \neq k$) and there is no parallel arithmetic overhead ($W_{\text{seq},i}(n,q) = \sum_{j=0}^{p-1} W_{P_j,i}(n,q)$). Let us solve $S_{\text{max}}(p,n,q)$ from (12) with $j = f$:

$$\begin{aligned} S_{\text{max}}(p,n,q) &= \frac{W_{\text{seq},i}(n,q)t_{w_f}}{W_{P_f,i}(n,q)t_{w_f}} = \frac{\sum_{j=0}^{p-1} W_{P_j,i}(n,q)}{W_{P_f,i}(n,q)} \\ &= \frac{\sum_{j=0}^{p-1} W_{P_f,i}(n,q) \frac{t_{w_f}}{t_{w_j}}}{W_{P_f,i}(n,q)} = \frac{1}{s_f} \end{aligned} \quad (14)$$

Hence the maximum speedup is the inverse of the fastest

processor normalized relative speed.

We can observe that using (10) and (6) in (12) the n_j values depend on the i iteration value, so the load balancing scheme is not static. A possible solution in order to get a suboptimum but static load balancing scheme is to get it for the worst case, where the load and the unbalancing can be higher: $i = m/q - 1$. Hence, we will begin getting n_{p-1} with $c_{0_{p-1}} = 1$, then n_{p-2} with $c_{0_{p-2}} = c_{0_{p-1}} + n_{p-1}$ and so on.

E. Communication analysis

The data that P_j must transfer to P_{j+1} for its i^{th} iteration (see figure 1) is the $\mathbf{L}_{(i)}$ matrix ($\frac{1}{2}q(q+1)$ elements), the nonzero part of \mathbf{M}_i ($q \sum_{k=0}^j n_k$ elements), the nonzero part of \mathbf{N}_i ($[i+1]q$ elements), and the \mathbf{H}_i matrix (qn elements). Let us suppose that a linear model fits the behavior of the communication time from P_j to P_{j+1} , then this time for the i^{th} iteration can be expressed as:

$$T_{C,P_j,i}(n, q) = \beta + \tau \left[qn + \frac{1}{2}q(q+1) + q \sum_{k=0}^j n_k + [i+1]q \right]$$

where β is the communication settling time and τ is the time to transfer one element.

We can consider two communication network models: one in which the transfer between adjacent processor can be made simultaneously (model A) and another in which these transfers must be done serially (model B).

For the model A, the communication time for the i^{th} iteration is:

$$\begin{aligned} T_{C,i}^{(A)}(n, q, p) &= \max_{j=0, \dots, p-2} \{T_{C,P_j,i}(n, q)\} \\ &= T_{C,P_{p-2}}(n, q) \\ &= \beta + \tau \left[qn + \frac{1}{2}q(q+1) + q(n - n_{p-1}) \right. \\ &\quad \left. + (i+1)q \right] \end{aligned}$$

If we ignore the pipeline filling or emptying time, the total communication time is:

$$\begin{aligned} T_C^{(A)}(m, n, q, p) &= \sum_{i=0}^{m/q-1} T_{C,i}(n, q, p) \\ &= \beta \frac{m}{q} + m\tau \left[n + \frac{1}{2}(q+1) \right. \\ &\quad \left. + (n - n_{p-1}) + \frac{1}{2} \cdot \frac{m}{q} + \frac{1}{2} \right] \\ &= \Theta(mn) + \Theta\left(\frac{m^2}{q}\right) \end{aligned} \quad (15)$$

For the model B:

$$T_{C,i}^{(B)}(n, q, p) = \sum_{j=0}^{p-2} T_{C,P_j,i}(n, q)$$

$$\begin{aligned} &= \sum_{j=0}^{p-2} \left\{ \beta + \tau \left[qn + \frac{1}{2}q(q+1) + q \sum_{k=0}^j n_k + (i+1)q \right] \right\} \\ &= (p-1) \left\{ \beta + \tau \left[qn + \frac{1}{2}q(q+1) + (i+1)q \right] \right\} \\ &\quad + \tau q \sum_{j=0}^{p-2} \sum_{k=0}^j n_k \\ &< (p-1) \left\{ \beta + \tau \left[qn + \frac{1}{2}q(q+1) + (i+1)q \right] \right\} \\ &\quad + \tau q(p-1)n \end{aligned}$$

And for all the iterations:

$$\begin{aligned} T_C^{(B)}(m, n, q, p) &< \\ &< \sum_{i=0}^{m/q-1} \left((p-1) \left\{ \beta + \tau \left[qn + \frac{1}{2}q(q+1) + (i+1)q \right] \right\} \right. \\ &\quad \left. + \tau q(p-1)n \right) \\ &< \frac{m}{q} \left((p-1) \left\{ \beta + \tau \left[qn + \frac{1}{2}q(q+1) + \frac{1}{2}m + \frac{1}{2}q \right] \right\} \right. \\ &\quad \left. + \tau q(p-1)n \right) \\ &< \frac{m}{q} (p-1)\beta + m(p-1)\tau \left[n + \frac{1}{2}(q+1) + \frac{1}{2q}m + \frac{1}{2} \right] \\ &\quad + \tau m(p-1)n \\ &= \Theta(mpn) + \Theta\left(\frac{m^2 p}{q}\right) \end{aligned} \quad (16)$$

F. Scalability analysis

We got null arithmetic overhead in the parallelization, so the overhead time in the parallel algorithm is only due to the communication time. We use the scalability concepts shown in [11], then in order to get the necessary problem size increment to keep constant a reference efficiency, we must compare the serial time with the total communication time overhead ($pT_C(m, n, q, p)$).

For the communication model A: $n = \Theta(p)$, $m = \Theta(p)$ and $n^2/m = \Theta(p/q)$ (if $m = \Theta(n)$, then $n = \Theta(p/q)$), so in general, $n = \Theta(p)$ and $m = \Theta(p)$

For the communication model B: $n = \Theta(p^2)$, $m = \Theta(p^2)$, and $n^2/m = \Theta(p^2/q)$ (if $m = \Theta(n)$, then $n = \Theta(p^2/q)$), so in general $n = \Theta(p^2)$ and $m = \Theta(p^2)$.

The scalability of the parallel system will range between $n, m = \Theta(p)$ and $n, m = \Theta(p^2)$, depending on the underlying network characteristics.

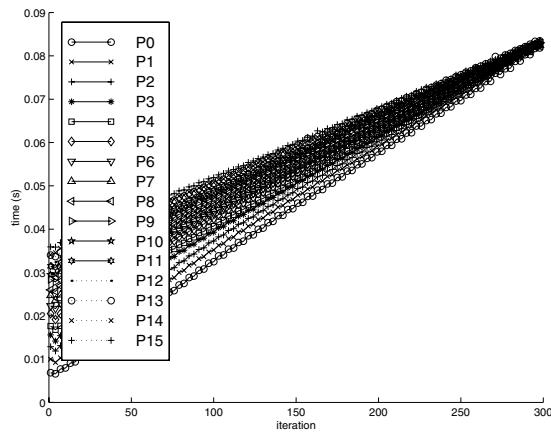


Fig. 2. Arithmetic time per iteration in every processor for $m = 6000$ and $q = 20$

G. Experimental results

The tests have been run in a ccNUMA architecture multiprocessor running a 64 bit Linux operating system with up to 16 processors available to one user. Each processor is a 1.3 GHz Itanium 2. The multiprocessor is organized as a hypercube in which every node is made up of two sets of two processors. The communication bandwidth between any two processors depends on the ubication of them in the system. The programs have been coded in Fortran using a proprietary MPI communications library. All the test were made with the parameter value $m = 6000$.

As an example of heterogeneous behavior, we have emulated a heterogeneous parallel system repeating the operations of every iteration only in some processors. With this code repetition technique, the heterogeneity is independent of the value of n . In the tests, the number of repetitions were two in the half of the processors.

The result of the load balancing can be observed in figure 2 for 16 processors. The figure depicts the arithmetic time per iteration (there are m/q iterations, where $q = 20$) and the load balancing based on (12) is got in the last iterations as expected.

The maximum speedup in this emulated heterogeneous parallel system is $3/4p$ (see (13) and (14)), so the maximum efficiency is 75%. Figure 3 depicts the experimental efficiency value of the heterogenous parallel system for 2, 4, 8 and 16 processors and $q = 20$, versus n .

IV. CONCLUSIONS

We propose a pipelined parallel algorithm to solve part of the OSIC decoding problem based on the square root Kalman Filter. All the processes derived from the parallel algorithm are regular so the execution in a heterogeneous network implies that the load must be distributed according to the processors speed. Although the ideal load balancing scheme is dynamic, the behavior of the static load balancing scheme in the heterogeneous system was satisfactory, with good efficiency results near to optimum values.

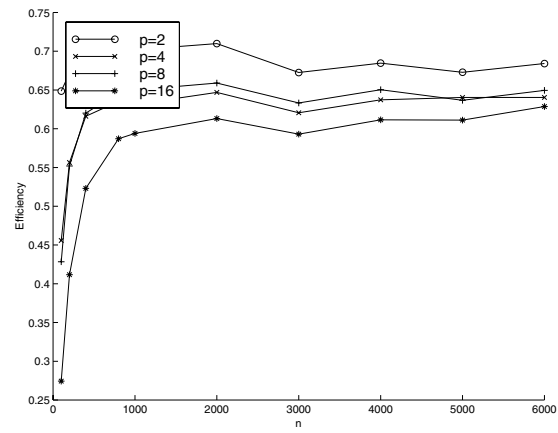


Fig. 3. Efficiency of parallel SRKF-OSIC for $m = 6000$ and $q = 20$

Acknowledgements.: This work has been supported by Spanish MEC and FEDER under grant TIC 2003-08238-C02 and by the Consejería de Educación de la Comunidad de Murcia, Fundación Séneca, project number 02973/PI/05.

REFERENCES

- [1] G.J. Foschini. Layered space-time architecture for wireless communications in a fading environment when using multiple antennas. *Bell Labs Technical Journal*, 1:41–59, 1996.
- [2] S. Haykin and M. Moher. *Modern Wireless Communications*. Prentice-Hall, Inc., NJ, USA, 2004.
- [3] M. Sellathurai and S. Haykin. Turbo-BLAST for wireless communications: theory and experiments. *IEEE Transactions on Signal Processing*, 50(10):2538–2546, Oct. 2002.
- [4] P.W. Wolniansky, G.J. Foschini, G.D. Golden, and R.A. Valenzuela. V-BLAST: An architecture for realizing very high data rates over the rich-scattering wireless channel. In *Proc. IEEE ISSSE-98*, pages 295–300, 1998.
- [5] E. Viterbo and J. Boutros. A universal lattice decoder for fading channels. *IEEE Trans. Inf. Theory*, 45(5), July 1999.
- [6] B. Hassibi. An efficient square-root algorithm for BLAST. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 2, pages II737 – II740, 2000.
- [7] Hufei Zhu, Zhongding Lei, and Francois P.S. Chin. An improved square-root algorithm for BLAST. *IEEE Signal Processing Letters*, 11(9), September 2004.
- [8] Yang-Seok Choi, Peter J. Voltz, and Frank A. Cassara. On channel estimation and detection for multicarrier signals in fast and selective Rayleigh fading channels. *IEEE Transactions on Communications*, 49(8), August 2001.
- [9] Ali H. Sayed and Thomas Kailath. A state-space approach to adaptive RLS filtering. *IEEE Signal Processing Magazine*, 11(3):18–60, July 1994.
- [10] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.
- [11] Vipin Kumar, Ananth Gram, Anshul Gupta, and George Karypis. *An Introduction to Parallel Computing: Design and Analysis of Algorithms*, chapter 4. Addison-Wesley, Harlow, England, second edition, 2003.