**The Journal of Supercomputing manuscript No.**
(will be inserted by the editor)

# Heterogeneous Pipelined square root Kalman Filter Algorithm for the MMSE-OSIC problem

**F.J. Martínez-Zaldívar · A.M. Vidal-Maciá · D. Giménez**

Received: date / Accepted: date

**Abstract** This paper describes a pipelined parallel algorithm for the MMSE-OSIC decoding procedure proposed in V-BLAST wireless MIMO systems, for heterogeneous networks of processors. It is based on a block version of the square root Kalman Filter algorithm that was initially devised to solve the RLS problem. It has been parallelized in a pipelined way obtaining a good efficiency and scalability. The optimum load balancing for this parallel algorithm is dynamic, but we derive a static load balancing scheme with good performance.

**Keywords** MMSE · OSIC · heterogeneous · V-BLAST · MIMO · pipeline · Kalman

## 1 Introduction

Multiple Input Multiple Output (MIMO) systems have been extensively studied in the context of wireless communications in 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 goal of increasing the capacity and reliability of the links. We focus our interest on the suboptimal but practical V-BLAST family where nearly optimal decoders such as MMSE (Minimum Mean Square Error Estimation) and its ordered version OSIC (Ordered Successive Interference Cancellation), [2,3], can be used. There are applications that use these methods to decode information in a digital receiver at a higher rate or with less bit error ratio than in SISO (Single Input Single Output) channels [4,5], where the dimension of the problem may be several

F.J. Martínez-Zaldívar
Dept. de Comunicaciones, Universidad Politécnica de Valencia, (Spain), E-mail: fjmartin@dcom.upv.es

A.M. Vidal-Maciá
Dept. de Sistemas Informáticos y Computación, U.P. de Valencia, (Spain), E-mail: avidal@dsic.upv.es

D. Giménez
Dept. de Informática y Sistemas, Universidad de Murcia, (Spain), E-mail: domingo@um.es

thousands, i.e. multicarrier systems such as OFDM used in DVB-T (Digital Video Broadcasting-Terrestrial), WIFI 802.11a, etc.

This paper describes the parallelization of the algorithm that solves the MMSE-OSIC decoding problem for a heterogeneous network of processors. The interest of the heterogeneous perspective in these applications is focused on future experimentations with GPUs where the CPU (with less computing power) collaborates in the problem solution. The algorithm is based on the ideas of [2] and can be used with the improvement presented in [3].

The paper is organized as follows: first we will describe the MMSE-OSIC decoding algorithm; next, we will describe the pipelined parallelization of this algorithm explaining the details of the proposed load balancing scheme; and finally, results of the experimentation in a heterogeneous parallel system will be stated.

## 2 MMSE-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 $\mathbf{H} \in \mathbb{C}^{m\times n}$, $m \geq n$, 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. Let us define $\mathbf{H}_\alpha$ as the *augmented channel matrix*:

$$\mathbf{H}_\alpha = \begin{pmatrix} \mathbf{H} \\ \sqrt{\alpha}\mathbf{I}_n \end{pmatrix}. \tag{1}$$

The use of MMSE (Minimum Mean Square Error estimation) yields [2]

$$\hat{\mathbf{x}} = \left\lfloor \begin{pmatrix} \mathbf{H} \\ \sqrt{\alpha}\mathbf{I}_n \end{pmatrix}^{\dagger} \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix} \right\rceil = \left\lfloor \mathbf{H}_\alpha^{\dagger|m}\mathbf{y} \right\rceil, \tag{2}$$

where $\hat{\mathbf{x}}$ is the estimation of $\mathbf{x}$, $\lfloor\cdot\rceil$ denotes the mapping or slicing of the result in the discrete symbol set, $\mathbf{H}_\alpha^{\dagger|m}$ denotes the first $m$ columns of the pseudoinverse of the *augmented* channel matrix (1), $\alpha^{-1}$ is a signal-to-noise ratio, and the asterisk superscript $(\cdot)^*$ denotes the complex conjugate. In OSIC, the signal components $x_i$, $i = 1, \ldots n$, are decoded from the strongest one (with the highest signal-to-noise ratio) to the weakest one, cancelling the contribution of the decoded signal component to the received signal, and then repeating the process with the remaining signal components, [2,6].

Every time the maximum signal-to-noise ratio remaining component is decoded it is necessary to compute the pseudoinverse. In [2,6] it is shown how to avoid this recomputation and how to optimize the way to obtain every component. Hence, we can obtain:

$$\mathbf{H}_\alpha^{\dagger|m} = \mathbf{P}^{1/2}\mathbf{Q}_\alpha^*, \tag{3}$$

where $\mathbf{P}^{1/2}$ is a square root factor of the error estimation covariance matrix $\mathbf{P} = \mathrm{E}\{(\mathbf{x}-\hat{\mathbf{x}})(\mathbf{x}-\hat{\mathbf{x}})^*\} = (\alpha\mathbf{I}_n + \mathbf{H}^*\mathbf{H})^{-1}$, and $\mathbf{Q}_\alpha \in \mathbb{C}^{m\times n}$ are the first $m$ rows of the orthogonal matrix $\mathbf{Q}$ from the *augmented* channel matrix QL-factorization,

$\mathbf{H}_\alpha = \mathbf{QL}$. The $\mathbf{P}^{1/2}$ and $\mathbf{Q}_\alpha$ matrices are the only necessary data to compute in order to obtain the result of the combined MMSE-OSIC algorithm. We can use the square root Kalman Filter algorithm [7] in order to obtain these matrices [2,6].

## 2.1 The square root Kalman Filter for MMSE-OSIC

From (3), $\mathbf{Q}_\alpha = \mathbf{H}_\alpha^{\dagger|m*}\mathbf{P}^{-*/2}$. This matrix is propagated throughout the iterations of the square root Kalman Filter that was initially devised to solve a RLS (Recursive Least Squares) problem. Next, we reproduce a block version of the algorithm for MMSE-OSIC that was reported in [2], which we refer to as SRKF-OSIC.

```
Input: $\mathbf{H} = \left(\mathbf{H}_0^*, \mathbf{H}_1^*, \ldots, \mathbf{H}_{m/q-1}^*\right)^*$, $\mathbf{P}_{(0)}^{1/2} = \frac{1}{\sqrt{\alpha}}\mathbf{I}_n$ 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 $i = 0, \ldots, m/q-1$ do
    Calculate $\mathbf{\Theta}_{(i)}$ and apply it in such a way that:
      $\mathbf{E}_{(i)}\mathbf{\Theta}_{(i)} = \begin{pmatrix} \mathbf{I}_q & \mathbf{H}_i\mathbf{P}_{(i)}^{1/2} \\ \mathbf{0} & \mathbf{P}_{(i)}^{1/2} \\ -\mathbf{\Gamma}_{(i+1)} & \mathbf{Q}_{\alpha,(i)} \end{pmatrix}\mathbf{\Theta}_{(i)} = \begin{pmatrix} \mathbf{R}_{e,(i)}^{1/2} & \mathbf{0} \\ \overline{\mathbf{K}}_{p,(i)} & \mathbf{P}_{(i+1)}^{1/2} \\ \mathbf{Z}_{(i)} & \mathbf{Q}_{\alpha,(i+1)} \end{pmatrix} = \mathbf{F}_{(i)}$
  end for
```

where $\mathbf{Z}_{(i)} = -\left(\mathbf{\Gamma}_{(i+1)}^* - \mathbf{H}_i\mathbf{H}_{\alpha,(i+1)}^{\dagger|m}\right)^* \mathbf{R}_{e,(i)}^{-*/2}$; $q$ is the number of consecutive rows of $\mathbf{H}$ processed in a block, thus $\mathbf{H}_i \in \mathbb{C}^{q\times n}$ is the $i^{\text{th}}$ block of $q$ consecutive rows of $\mathbf{H}$; the iteration index subscript enclosed between parenthesis denotes that the variable is updated iteratively, i.e., $\mathbf{Q}_\alpha$ and $\mathbf{P}^{1/2}$ are the values $\mathbf{Q}_{\alpha,(i+1)}$ and $\mathbf{P}_{(i+1)}^{1/2}$ in the last iteration $i = m/q - 1$. $\mathbf{\Gamma}_{(i+1)} = \left(\mathbf{0}_{iq\times q}^T, \mathbf{I}_q, \mathbf{0}_{(m-q(i+1))\times q}^T\right)^T \in \mathbb{R}^{m\times q}$. $\mathbf{R}_{e,(i)}$ and $\overline{\mathbf{K}}_{p,(i)}$ are variables of the Kalman Filter whose meanings are described in [7], and $\mathbf{H}_{\alpha,(i+1)}^{\dagger|m}$ appears implicitly in $\mathbf{Z}_{(i)}$.

The cost of one iteration is a matrix multiplication $\mathbf{H}_i\mathbf{P}_{(i)}^{1/2}$ and the application of a sequence of Givens rotations $\mathbf{\Theta}_{(i)}$, exploiting and maintaining the triangular structure of $\mathbf{P}_{(i)}^{1/2}$ throughout the iterations. The total cost of the iterations is $W_{\text{seq}}(m,n,q) = 4n^2m + 3nm^2$ flops approximately.

## 3 Parallel algorithm

A matrix enclosed within square brackets with a processor subscript denotes that part of the matrix belongs to that processor. If it is enclosed within parenthesis, then it denotes that the entire matrix is in the processor of the subscript.

The last $n$ columns of either $\mathbf{E}_{(i)}$ or $\mathbf{F}_{(i)}$ matrices of the SRKF-OSIC algorithm will be distributed among the $p$ processors: $P_j$ will own $n_j$ columns beginning at $c_{0_j}$ with $n = \sum_{j=0}^{p-1} n_j$. The first $q$ columns of either $\mathbf{E}_{(i)}$ or $\mathbf{F}_{(i)}$ will be denoted by $\mathbf{D}_{(i)} \in \mathbb{C}^{(q+n+m)\times q}$. $\mathbf{D}_{(i)}$ will be manipulated in a pipelined way by all the processors, so we denote it as $(\mathbf{D}_{(i)})_{P_j}$, where $P_j$ represents the processor that is processing it. $\mathbf{D}_{(i)}$ is made up of a lower triangular matrix $\mathbf{J}_{(i)} \in \mathbb{C}^{q\times q}$ and two general matrices $\mathbf{M}_{(i)} \in \mathbb{C}^{n\times q}$ and $\mathbf{N}_{(i)} \in \mathbb{C}^{m\times q}$. $\mathbf{M}_{(i)}$ will be divided into $p$ groups of rows, (the number of rows will be indicated by a left superscript). The nonzero rows of $\mathbf{N}_{(i)}$ will depend on the iteration index $i$.

## 3.1 Processor tasks

When the $P_j$ processor obtains zeroes in $[\mathbf{H}_i\mathbf{P}^{1/2}_{(i)}]_{P_j}$ for the $i$th iteration, the matrices $[\mathbf{P}^{1/2}_{(i)}]_{P_j}$ and $[\mathbf{Q}_{\alpha,(i)}]_{P_j}$ are converted into $[\mathbf{P}^{1/2}_{(i+1)}]_{P_j}$ and $[\mathbf{Q}_{\alpha,(i+1)}]_{P_j}$ respectively. Therefore, if $P_j$ has obtained $\mathbf{H}_{i+1}$, it can obtain $[\mathbf{H}_{i+1}\mathbf{P}^{1/2}_{(i+1)}]_{P_j}$ and begin the process for the $(i+1)$th iteration. The pipelined behaviour of the algorithm is based on this fact.

As an example, let us suppose that we have $p = 3$ processors: $P_0$, $P_1$ and $P_2$. In $P_0$, the first step in the pipelined parallel algorithm is presented below (an apostrophe denotes the updating of the variable):

$$\mathbf{E}'_{(i)} = \mathbf{E}_{(i)}\mathbf{\Theta}_{i,P_0}$$

$$= \left( \begin{pmatrix} \mathbf{I}_q \\ {}^{n_2}[\mathbf{0}] \\ {}^{n_1}[\mathbf{0}] \\ {}^{n_0}[\mathbf{0}] \\ -\mathbf{\Gamma}_{(i+1)} \end{pmatrix}_{P_0} \begin{bmatrix} \mathbf{H}_i\mathbf{P}^{1/2}_{(i)} \\ \mathbf{P}^{1/2}_{(i)} \\ \mathbf{Q}_{\alpha,(i)} \end{bmatrix}_{P_2} \begin{bmatrix} \mathbf{H}_i\mathbf{P}^{1/2}_{(i)} \\ \mathbf{P}^{1/2}_{(i)} \\ \mathbf{Q}_{\alpha,(i)} \end{bmatrix}_{P_1} \begin{bmatrix} \mathbf{H}_i\mathbf{P}^{1/2}_{(i)} \\ \mathbf{P}^{1/2}_{(i)} \\ \mathbf{Q}_{\alpha,(i)} \end{bmatrix}_{P_0} \right) \mathbf{\Theta}_{i,P_0}$$

$$= \left( \begin{pmatrix} \mathbf{J}_{(i)} \\ {}^{n_2}[\mathbf{0}] \\ {}^{n_1}[\mathbf{0}] \\ {}^{n_0}[\mathbf{M}_{(i)}] \\ \mathbf{N}_{(i)} \end{pmatrix}_{P_0} \begin{bmatrix} \mathbf{H}_i\mathbf{P}^{1/2}_{(i)} \\ \mathbf{P}^{1/2}_{(i)} \\ \mathbf{Q}_{\alpha,(i)} \end{bmatrix}_{P_2} \begin{bmatrix} \mathbf{H}_i\mathbf{P}^{1/2}_{(i)} \\ \mathbf{P}^{1/2}_{(i)} \\ \mathbf{Q}_{\alpha,(i)} \end{bmatrix}_{P_1} \begin{bmatrix} \mathbf{0} \\ \mathbf{P}^{1/2}_{(i+1)} \\ \mathbf{Q}_{\alpha,(i+1)} \end{bmatrix}_{P_0} \right),$$

where $\mathbf{\Theta}_{i,P_0}$ is a unitary transformation calculated and applied by $P_0$ in order to obtain zeroes in $[\mathbf{H}_i\mathbf{P}^{1/2}_{(i)}]_{P_0}$. Now, $P_0$ must transfer the nonzero part of $(\mathbf{D}_{(i)})_{P_0}$ to $P_1$. If $P_0$ has received $\mathbf{H}_{i+1}$, it has all the necessary data to make up $[\mathbf{H}_{i+1}\mathbf{P}^{1/2}_{(i+1)}]_{P_0}$ and start again. When $P_1$ receives $(\mathbf{D}_{(i)})_{P_0}$, then:

$$\mathbf{E}''_{(i)} = \mathbf{E}'_{(i)}\mathbf{\Theta}_{i,P_1}$$

$$= \left( \begin{pmatrix} \mathbf{J}'_{(i)} \\ {}^{n_2}(\mathbf{0}) \\ {}^{n_1}[\mathbf{M}_{(i)}] \\ {}^{n_0}[\mathbf{M}_{(i)}]' \\ \mathbf{N}'_{(i)} \end{pmatrix}_{P_1} \begin{bmatrix} \mathbf{H}_i\mathbf{P}^{1/2}_{(i)} \\ \mathbf{P}^{1/2}_{(i)} \\ \mathbf{Q}_{\alpha,(i)} \end{bmatrix}_{P_2} \begin{bmatrix} \mathbf{0} \\ \mathbf{P}^{1/2}_{(i+1)} \\ \mathbf{Q}_{\alpha,(i+1)} \end{bmatrix}_{P_1} \begin{bmatrix} \mathbf{0} \\ \mathbf{P}^{1/2}_{(i+1)} \\ \mathbf{Q}_{\alpha,(i+1)} \end{bmatrix}_{P_0} \right).$$

The comments given above for $P_0$ also hold for $P_1$. When $P_2$ receives the nonzero part of $(\mathbf{D}_{(i)})_{P_1}$, then:

$$\mathbf{E}'''_{(i)} = \mathbf{E}''_{(i)}\mathbf{\Theta}_{i,P_2}$$

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

$$= \begin{pmatrix} \mathbf{R}^{1/2}_{e,(i)} & \mathbf{0} \\ \overline{\mathbf{K}}_{p,(i)} & \mathbf{P}^{1/2}_{(i+1)} \\ \mathbf{Z}_{(i)} & \mathbf{Q}_{\alpha,(i+1)} \end{pmatrix} = \mathbf{F}_{(i)}.$$

The comments given above for $P_0$ and $P_1$ also hold for $P_2$. (See Figure 1 which depicts the behavior of the parallel algorithm when $P_0$ is processing the $i^{\text{th}}$ iteration.)

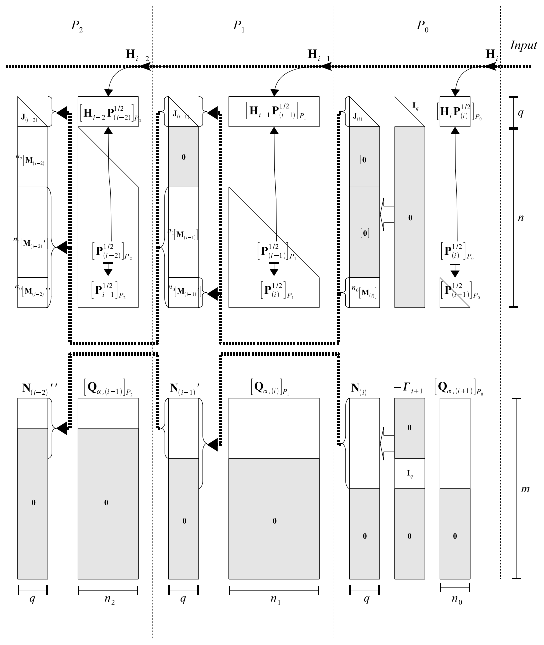

**Fig. 1** SRKF-OSIC pipelined parallel algorithm.

The arithmetic cost in the $i^{\text{th}}$ iteration in the $P_j$ processor is due to the matrix multiplication $[\mathbf{H}_i\mathbf{P}^{1/2}_{(i)}]_{P_j}$ and to the zeroing in the $n_j$ columns of $[\mathbf{H}_i\mathbf{P}^{1/2}_{(i)}]_{P_j}$ (columnwise and from right to left). We can verify that the parallelization arithmetic overhead is null.

## 3.2 Load balancing in heterogeneous networks of processors

The perfect load balance is obtained when all the processors require the same amount of time to process their assigned workload in certain time interval. In the pipelined algorithm each processor processes a different iteration at the same time instant. It is difficult to get an analytical expression to balance the load with this consideration. We can consider a simpler criterion to balance the workload:

$$W_{P_j,i}(n,q)t_{w_j} = W_{P_k,i}(n,q)t_{w_k},\ \forall j \neq k \tag{4}$$

where $t_{w_j}$ and $t_{w_k}$ are the time per flop in $P_j$ and $P_k$ respectively. The $n_j$, $\forall j$, values can be obtained by solving the following ideal equation:

$$W_{P_j,i}(n,q)t_{w_j} = \frac{W_{\text{seq},i}(n,q)t_{w_{\text{seq}}}}{S_{\max}(p,n,q)},\ \ \forall\, 0 \leq j \leq p-1, \tag{5}$$

where $S_{\max}(p,n,q)$ is the maximum speedup attainable in the parallel system and $t_{w_{\text{seq}}}$ is the time per flop for the sequential algorithm.

The maximum speedup in the heterogeneous network depends on the time per flop $t_{w_j}$ of each processor. Let us define $s_j$ as the normalized relative speed of the processors (dimensionless):

$$s_j = \frac{1}{\sum_{r=0}^{p-1} \frac{tw_j}{t_{w_r}}}. \tag{6}$$

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

Let us suppose that the sequential algorithm is run on the fastest processor of the heterogeneous network, say $P_f$, $0 \leq f \leq p-1$, then $t_{w_{\text{seq}}} = t_{w_f}$. The maximum speedup can be obtained from (5) when the load balance is got (4) and there is no parallel arithmetic overhead (subsection 3.1). Let us solve $S_{\text{max}}(p,n,q)$ from (5) with $j = f$:

$$S_{\max}(p,n,q) = \frac{W_{\text{seq},i}(n,q)t_{w_f}}{W_{P_f,i}(n,q)t_{w_f}} = \frac{\sum_{j=0}^{p-1} W_{P_f,i}(n,q)\frac{t_{w_f}}{t_{w_j}}}{W_{P_f,i}(n,q)} = \frac{1}{s_f}. \tag{7}$$

Hence, the maximum speedup in the heterogeneous network is the inverse of the normalized relative speed of the fastest processor.

The (sub)optimal $n_j$ values depend on the $i$ iteration value, so the proposed load balancing scheme is not static (hence, we should change the column distribution at every iteration). One possible solution to obtain a suboptimum but static load balancing scheme is to get it for the worst case, where the work load and the unbalancing is the highest (at the last iteration: $i = m/q - 1$). Hence, we will begin by obtaining $n_{p-1}$ 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.

### 3.3 Communication analysis

The data that $P_j$ must transfer to $P_{j+1}$ for its $i^{\text{th}}$ iteration (see Figure 1) is the nonzero part of $(\mathbf{D}_{(i)})_{P_j}$ and the $\mathbf{H}_i$ submatrix. Let us suppose that a linear model fits this communication time. We can consider two communication network models: one in which the transfer between adjacent processors can be made simultaneously (model A, i.e., a linear array topology) and another in which these transfers must be done serially (model B, i.e., bus topology). Hence, the total communication time for both models are $\mathbf{\Theta}\left(mn\right) + \mathbf{\Theta}\left(m^2/q\right)$ and $\mathbf{\Theta}\left(mnp\right) + \mathbf{\Theta}\left(m^2/qp\right)$ respectively.

### 3.4 Scalability analysis

We obtained a null arithmetic overhead in the parallelization, so the overhead time in the parallel algorithm is only due to the communication time. The serial time must be compared with the total communication time overhead in order to get the scalability of the parallel system [8]. In our case, the scalability ranges between $n, m = \mathbf{\Theta}\left(p\right)$ and $n, m = \mathbf{\Theta}\left(p^2\right)$ for the A and B models respectively. Hence, a linear scalability behavior can be obtained with just a linear array topology.

### 3.5 Experimental results

The heterogeneous system used to test the parallel algorithm consisted of:

– Node 0: A monoprocessor with a 1.7 GHz Pentium IV with 1 GB of main memory, running a CentOS 4.4 32-bit operating system.
– Node 1: a CC-NUMA multiprocessor with two 1.4 GHz two-cores Itanium processors with 8 GB of memory running a Red Hat Enterprise Linux AS 64-bit operating system.

using the MPI library (MPICH 1.2.7p1), compiled for 32 and 64 bits, respectively.

Figure 2 depicts the arithmetic time per iteration ($i = 0, \ldots, m/q - 1$) with the proposed static balancing scheme observing that it is got in the last iterations as expected.

Taking as reference times the execution time of the sequential algorithm in both kinds of processors, the normalized relative speed of the processors (6) were $s_0 \approx 0.04$ and $s_1 = \ldots = s_4 \approx 0.24$, for $m = 6000$ and $q = 20$ and independent from $n$. The maximum normalized relative speed in this system is $\max\{s_j\} \approx 0.24$, then $S_{\max} \approx 4.2$. Hence, the maximum attainable efficiency is $E_{\max} = S_{\max}/p = 4.2/5 \approx 83\%$. Figure 3 depicts the efficiency of the parallel algorithm. For high values of $n$, the efficiency is about 60–65%, which represents a *relative* efficiency of 72–78% respect to the maximum.

## 4 Conclusions

We have proposed a pipelined parallel algorithm to solve the heaviest part of the MMSE-OSIC decoding problem based on the square root Kalman Filter algorithm.

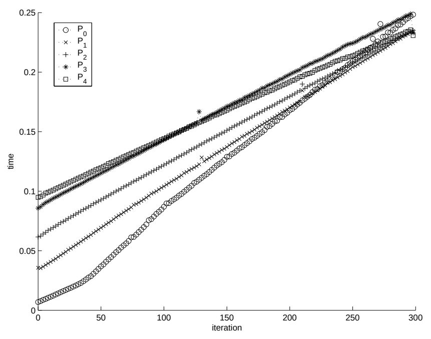

**Fig. 2** Arithmetic time (s) per iteration in each processor for $m = 6000$, $q = 20$, and $n = 2000$.

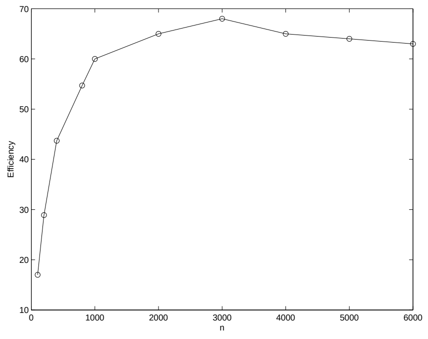

**Fig. 3** Efficiency (%) of parallel SRKF-OSIC for $m = 6000$ and $q = 20$.

All the processes derived from the parallel algorithm are regular, so the execution in a heterogeneous network implies that the load must be distributed evenly according to the speed of the processors. Although the ideal load balancing scheme for our parallel algorithm is dynamic, the behavior of the proposed static load balancing scheme in the heterogeneous system was satisfactory, with good efficiency results near to optimum values. The algorithm parallelization can be used in MIMO detection applications with GPU, FPGA or other VLSI systems implementations due to its implicit regularity.

**Acknowledgements** This work has been supported by the Generalitat Valenciana, project 20080811, by the Universidad Politécnica de Valencia, project 20080009, by the Conserjería de Educación de la Región de Murcia (Fundación Séneca, 08763/PI/08), and by the Ministerio de Ciencia e Innovación (TIN2008-06570-C04-02).

## 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. 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.
3. Hufei Zhu and Zhongding Lei and Francois P.S. Chin. An improved square-root algorithm for BLAST. *IEEE Signal Processing Letters*, 11(9), September 2004.
4. Yang-Seok Choi and 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.
5. Burg, A. and Haene, S. and Perels, D. and Luethi, P. and Felber, N. and Fichtner, W. Algorithm and VLSI architecture for linear MMSE detection in MIMO-OFDM systems. In *Proceedings of the IEEE Int. Symp. on Circuits and Systems*, May 2006.
6. F.J. Martínez Zaldívar. *Algoritmos paralelos segmentados para los problemas de Mínimos Cuadrados Recursivos (RLS) y de Detección por Cancelación Ordenada y Sucesiva de Interferencia (OSIC)*. PhD thesis, Facultad de Informática, Universidad Politécnica de Valencia (Spain), 2007.
7. Ali H. Sayed and Thomas Kailath. A state-space approach to adaptive RLS filtering. *IEEE Signal Processing Magazine*, 11(3):18–60, July 1994.
8. Vipin Kumar and Ananth Gram and Anshul Gupta and George Karypis. *An Introduction to Parallel Computing: Design and Analysis of Algorithms*, chapter 4. Addison-Wesley, Harlow, England, second edition, 2003.