Parallel algorithms and architectures for subspace based channel estimation for CDMA communication systems

Chaitali Sengupta, Kishore Kota and Joseph R. Cavallaro,

Rice University, ECE Dept. P. O. Box 1892, Houston, TX 77251.
Ph:(713)-527-4719 Fax:(713)-524-5237

ABSTRACT

This paper presents an overview and results from an ongoing research project to study parallel algorithms for the acquisition of *Code Division Multiple Access* (CDMA) communication signals. The goal of this research is to evaluate a class of related algorithms and architectures for the acquisition problem and map them onto parallel architectures containing DSPs.

The algorithms used are generally termed *subspace-based algorithms*, since they involve computation of subspaces of the vector space spanned by certain observation vectors. This paper presents results from some preliminary implementations of such a subspace-based algorithm on the Texas Instruments TMS320C40 Parallel Processing Development System.

Keywords: CDMA, parallel algorithms and architectures

1. INTRODUCTION

The goal of this work is to develop parallel architectures for implementing algorithms for the estimation of the signal and channel parameters in Code Division Multiple Access (CDMA) communication systems. CDMA has emerged as an access protocol for a number of multiuser applications including cellular mobile radio networks. For the next generation of cellular CDMA networks, it will be important to reduce the near-far effect where the signals of strong users interfere with the demodulation of the signals from weak users. As an alternative to the use of power control, the multiple access interference that occurs can be dealt with through the use of novel signal processing techniques.

Subspace-based algorithms have previously been developed for direction finding problems and have only recently been extended to near-far resistant acquisition of CDMA signals.[1] These new algorithms for CDMA acquisition can be mapped onto a variety of parallel architectures, thus greatly improving system performance. This paper presents work on the mapping of these algorithms to the Texas Instruments TMS320C40 and the Parallel Processing Development System.

The CDMA acquisition problem is described in section 2. In section 3, parallel algorithms to solve the problem are reviewed. Section 4, reports the implementation of the subspace-based algorithm on the Parallel Processing Development System and an analysis of the results.

2. BACKGROUND

2.1. Code Division Multiple Access communications

In CDMA systems, a communication channel with a given bandwidth is accessed by all the users *simultaneously*. The different mobile users are distinguished at the base station receiver by the unique code used by the users to modulate their signals. Hence, the CDMA signal transmitted by any given user consists of that user's data which modulates a unique *spreading code* assigned to each user, which in turn modulates a carrier (the frequency of which is the same for all the users), using any well-known modulation scheme such as *binary phase shift keying (BPSK)*. The receiver receives a linear superposition of the signals transmitted by all the users, attenuated by arbitrary (possibly time-varying) factors and delayed by an arbitrary amount. The goal of acquisition is to determine these unknown attenuation factors and unknown delays by processing the received signal to facilitate recovery of the data transmitted by each user.

Channel estimation is one of the major problems in radio communications particularly when the mobile radio system is subject to multipath fading, multiple access interference, or intersymbol interference. When the system is asynchronous and coherent, the channel estimation problem also includes estimation of the delays and phases of the incoming signals. When the CDMA technique is used to allow multiple users access to a single channel, the system is susceptible to the so-called *near-far* effect. The near-far problem arises when the signals from the different users arrive at the receiver with widely varying power levels. The near-far problem has been shown to severely degrade the performance of matched filter detectors in conventional CDMA systems. Our studies indicate that similar degradation in performance occurs in conventional delay and phase acquisition and tracking systems. Therefore, in order to improve the performance of CDMA systems in the presence of the near-far effect, there is a need for the development of algorithms for fast acquisition and responsive tracking that are near-far resistant and effective when the channel has multiple propagation paths.

2.2. CDMA acquisition as a subspace-based algorithm

The class of algorithms, being studied, to perform near-far resistant acquisition is called *subspace-based techniques*. The formulation of the CDMA acquisition problem as a subspace-based algorithm appears in an earlier paper.[1]

A key to the application of these techniques is to find at any discrete observation time instant, i, a linear model of the observations in the form:

$$\mathbf{y}_i = \mathbf{A}\,\mathbf{c}_i + \boldsymbol{\eta}_i, \tag{1}$$

where $\mathbf{y}_i$ is an observation vector, $\mathbf{c}_i$ is the input to a linear model, $\boldsymbol{\eta}_i$ is an additive noise vector and $\mathbf{A}$ is the response of the linear system. In this model, the input to the system $\mathbf{c}_i$ changes with every observation. The system response $\mathbf{A}$ is a known (nonlinear) function of unknown parameters $\boldsymbol{\tau}$. The parameters $\boldsymbol{\tau}$, if they vary, are assumed to vary slowly and consequently the system response is approximately constant over several observations. Given this linear model, it is possible through a process of averaging to estimate the unknown parameters $\boldsymbol{\tau}$.

When the CDMA acquisition problem is formulated as a subspace-based algorithm, observations by the receiver may be written in this form, where the unknown parameters, $\boldsymbol{\tau}$, are the relative time delays of the users, the columns of $\mathbf{A}$ are functions of the users' spreading code and the unknown delay and $\mathbf{c}_i$ contain information about the different users' data and the arbitrary attenuation factors. For details of how the continuous time received signal is processed to result in the above discrete model, the reader is referred to the previously mentioned paper.[1]

2.3. Parameter estimation via subspace-based techniques

Given observation vectors $\mathbf{y}_i$ for all i, which may be modeled by Eq. 1, the first step is the estimation of an orthonormal basis for the subspace orthogonal to the subspace spanned by the columns of the matrix $\mathbf{A}$. The subspace spanned by the columns of $\mathbf{A}$ is called the *signal subspace* and the orthogonal subspace is termed *noise subspace*. This basis may be obtained by computing an eigenvalue decomposition of the correlation matrix of the observation vectors, where the correlation matrix $\mathbf{R}$ is defined as:

$$\mathbf{R} = \mathcal{E}[\mathbf{y}_i \mathbf{y}_i'], \tag{2}$$

where $\mathcal{E}[.]$ denotes the expected value and $'$ denotes the conjugate transpose operator. In practice, since this correlation matrix is not known, it is estimated as a time average of $\mathbf{y}_i \mathbf{y}_i'$ over several observations. Since this average may be performed in a variety of ways, several formulations of the problem result and in effect result in different algorithms, which in turn may be implemented in a variety of different ways. One of these formulations is the *block formulation*, where the observations are separated into *blocks* of L successive observations and the correlation matrix for the j^{th} block is estimated as:

$$\hat{\mathbf{R}}_j = \frac{1}{L} \sum_{i=j-L+1}^{j} \mathbf{y}_i \mathbf{y}_i' = \frac{1}{L} \mathbf{Y}_j' \mathbf{Y}_j. \tag{3}$$

The matrix $\mathbf{Y}_j$ is an observation matrix, which is defined as follows for the block formulation:

$$\mathbf{Y}_j = \begin{pmatrix} \mathbf{y}_{j-L+1}' \\ \mathbf{y}_{j-L+2}' \\ \cdots \\ \mathbf{y}_j' \end{pmatrix}. \tag{4}$$

The block size, L, should be chosen large enough to result in a reasonable estimate of the correlation matrix, while simultaneously being small enough that the time delays do not change appreciably within any given block. The alternative formulation of the estimated correlation matrix $\hat{\mathbf{R}}_j$ in terms of an observation matrix $\mathbf{Y}_j$ immediately suggests the use of a well-known identity from numerical linear algebra. The eigenvalue decomposition of the correlation matrix may be computed directly from the singular value decomposition (SVD) of the observation matrix. This provides other benefits which are not obvious at first glance. First, it obviates the need for the matrix multiply operation required to compute the correlation matrix. In addition, it reduces the wordlength requirements by half and allows pipelining of the entire algorithm.

Once an orthogonal basis for the noise subspace, $\mathcal{S}_N$, has been determined, the unknown time delays are estimated by solving non-linear optimization problems which may again be posed in numerous ways. One of these formulations is the *deterministic MUSIC*[13] formulation, which is based on the *Multiple Signal Classification (MUSIC)*[2] algorithm. In this formulation, the different users' time delays are estimated independently of each other. This is done by solving the following nonlinear-optimization problem for each user, k:

$$\hat{\tau}_k = \arg \min_{\tau_k \in [0, NT_c)} ||\mathbf{a}_k(\tau_k)' \mathcal{S}_N||_2^2, \tag{5}$$

where, τ_k is the time delay of user k and $\hat{\tau}_k$ is the estimate of that delay. Data is transmitted in terms of *bits*. Each bit is modulated using the spreading code, consisting of N *chips*. T_c represents the chip period. The vector $\mathbf{a}_k(\tau_k)$ is the spreading waveform for user k as a function of the time delay of that user. The formulation essentially estimates the time delay of user k as that value chosen from the valid range of time delays, which results in the vector with smallest components when projected into the estimated noise space.

3. PARALLEL ARCHITECTURES FOR CDMA ACQUISITION

The real-time implementation of the proposed algorithms, supporting cellular data rates in excess of 20 kilo-bits/sec, poses numerous challenges to current technology, requiring the development of efficient parallel algorithms and architectures. Numerous techniques, which have previously been developed for the direction of arrival problem in antenna array signal processing, find application in the CDMA acquisition problem. Recent interest in the direction of arrival problem has led to the development of several computationally efficient algorithms and architectures for many of our subproblems.

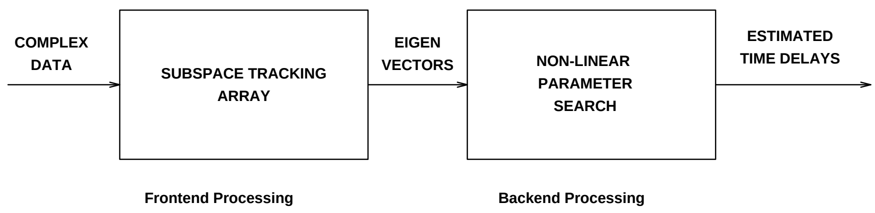


Figure 1: Block diagram of a subspace-based channel estimation system

Since the subspace-based algorithms require quite a lot of computation, it is convenient to separate them into smaller parts. An intuitive and convenient way to handle these algorithms is to split them as a frontend subspace decomposition and a backend non-linear optimization problem (Fig. 1). Other factors which guide such a treatment is that the subspace tracking problem is a well-defined linear algebra problem that has been studied extensively in the literature. The problem is characterized by regular computation ideally suited for parallel arrays. The frontend computation is almost completely independent of the algorithms used in the backend and often independent of the overall application.

The backend computation on the other hand is marked by irregular computation, where the execution times may not be predictable. Often a closed form solution is not feasible due to the multimodal nature of the objective functions, and an extensive search may be needed to solve the problem. The backend problem formulation is much more dependent on the overall problem. The above separation also allows development of several alternative designs for the frontend and backend, which can then be mixed and matched at will with very few restrictions.

3.1. Frontend algorithms

The functionality of the frontend consists of computing a singular value decomposition of the input observation matrix ($Y_j = U\Sigma V$), which is an $O(n^3)$ operation, where n defines the matrix dimensions. Certain computational reductions can be achieved if the backend requires only the singular vectors and not the singular values. Typically, it is not necessary to completely sort the singular vectors. It is sufficient to separate the signal and noise vectors. Further computational reductions can be achieved if only a few signal or noise vectors are required. Algorithms which compute an approximate decomposition can result in further reductions in error.

The algorithms developed for the frontend also depend on the way the subspace-based algorithms derived for stationary conditions, are extended to non-stationary conditions. As such the frontend algorithms can be separated into recursive and non-recursive (or block) formulations. The recursive algorithms may be used when an older decomposition is updated whenever new data is received. The non-recursive algorithms on the other hand compute the desired quantities from scratch. The recursive algorithms in general result in less computation per observation than the non-recursive techniques. Hence they are attractive when parameters do not change frequently and a lower cost solution is desired.

Table 1: *Classification of frontend algorithms*

Algorithm	comp/observation	Computed quantities	Parallelism
Non-recursive subspace tracking			
QR algorithm[3]	$O(n^2)$	Σ, V	–
Jacobi method[4]	$O(n^2 \log n)$	Σ, V	$O(n^2)$
Hestenes method[5]	$O(n^2 \log n)$	Σ, V	$O(n)$
One-sided Jacobi method[6]	$O(n^2 \log n)$	Σ, V	$O(n)$
Pipelined SVD[7]	$O(n^2 \log n)$	Σ, V	$O(n \log n)$
Recursive subspace tracking			
SVD updating[8]	$O(n^3 \log n)$	Σ, V	$O(n^2)$
Approximate SVD updating[9]	$O(n^2)$	Σ, V	$O(n^2)$

Many of these algorithms also have different degrees of parallelism. Hence, some of the algorithms are amenable to implementation on linear arrays of dimension proportional to the problem size, to result in similar throughput increases. Other algorithms exhibit even higher degrees of parallelism and allow implementation on arrays whose size is limited by the square of the dimension of the problem, with corresponding throughput increases. Several algorithms and architectures are possible for each formulation. A few of these algorithms are tabulated in Table 1.

3.2. Backend algorithms

There are a number of different formulations of the non-linear optimization problem in the backend. The better estimators usually require both singular vectors and singular values in the formulation of the optimization problem. This provides a trade-off between the complexity of the frontend process and the accuracy of the backend estimates.

In addition, the optimization problem also determines the amount of parallelism available in the objective function evaluation, an essential operation in non-linear optimization. For the CDMA acquisition problem, the use of the deterministic MUSIC estimator results in the optimization problem previously described by Eq. 5. This estimator allows considerable parallelism in the evaluation of the objective function. In addition, the problem lends itself to a closed form solution within each chip of the spreading code, which further simplifies design of a parallel architecture.

4. IMPLEMENTATION OF THE FRONTEND AND BACKEND ON THE TI PPDS

In this part of the experiment, the goal was to get a working prototype of a parallel system implementing both the frontend and the backend. For this purpose, the TI Parallel Processing Development System (PPDS) was chosen, as the PPDS and its related software gave a quick and efficient means of testing out the algorithms on an existing parallel system. The hardware platform used, consists of the TMS320C40 PPDS connected to a Sun workstation through a XDS510WS emulator.

The TMS320C40 is a parallel processing DSP. In addition to a powerful CPU that can execute up to 11 operations per cycle with a 40 to 50 ns cycle time, it contains 6 communication ports and a multichannel DMA.[10]The on-chip communication ports allow direct processor-to-processor communication, and the DMA unit provides concurrent I/O by running parallel to the CPU. The TMS320C40 PPDS has four on-board TMS320C40s, each supported by a local bus with $64K \times 32$-bit words of zero wait-state static RAM.[11] The 4 processors are fully connected and can hence be used to implement almost any parallel configuration. All these features make the TMS320C40 PPDS very suitable for implementation of the acquisition algorithm. Also, as this system could be used in a receiving base station, parallelization upto 4 processors is a reasonable hardware environment.

As discussed in,[12] the TMS320C40 has single-cycle functional units, but, several cycles are required to transfer data across the six communication ports. Thus, the PPDS is not particularly suited for fine grained parallel processing. This motivated the choice of the parallelization technique for both the frontend and the backend. If the PPDS was configured as a 2×2 array or a pipelined array handling only a few matrix elements or a single column at a time, the communication overhead compared to the computation time would be large. Hence, processor utilization would be low. Instead, the PPDS was configured as a linear array of 4 processors and the columns of the data matrix was distributed between the 4 processors. Each processor works independently on its own set of columns and then exchanges columns with other processors. Thus, this scheme implements coarse grained parallelism which gives speedups close to 4 and hence, high processor utilization.

4.1. Implementation of SVD (using the Hestenes method) on the PPDS

For the frontend, the Hestenes method, which is a non-recursive procedure was chosen. Such an algorithm should be used when the singular values and singular vectors are being computed from scratch. Once the SVD is obtained using such a non-recursive procedure, a recursive algorithm may be used later to update the decomposition whenever new data arrives.

The Hestenes method for SVD was parallelized as described by Schreiber.[5] The parallel architecture used is a linear array connected as shown in Fig. 2. This implementation is based on 4 processors. The number of processors remains fixed even though the problem size, that is the size of the input matrix, varies. The partitioning scheme in the algorithm takes care of distributing the problem on the fixed and undersized linear array. As discussed in section 2.3, the size of the input matrix which is decomposed, is $N \times L$, where N is the spreading code size and L is the block size.

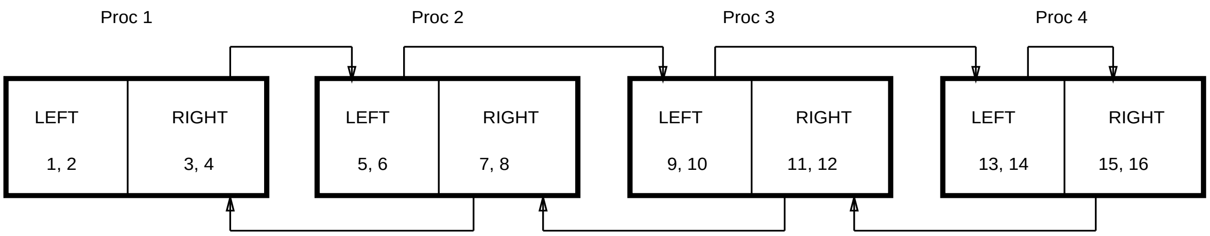


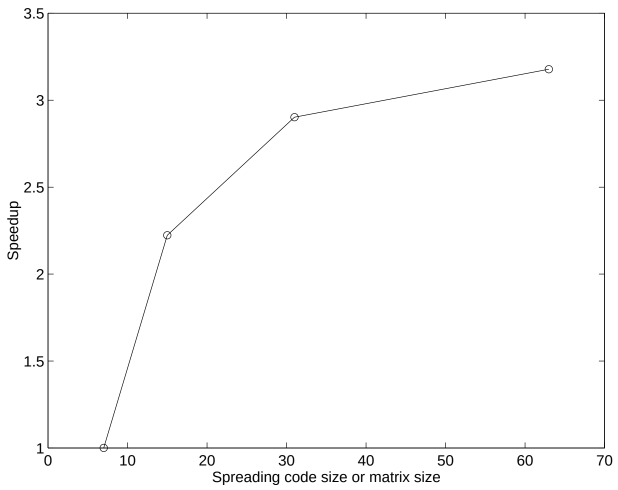
Figure 2: The parallel array used for the implementation of the SVD

For implementing the frontend, the PPDS was used as a distributed memory linear array with the 4 processors connected as shown in Fig. 2. The figure also shows how a matrix with 16 columns is distributed among the 4 processors. The data memory of each processor is conceptually divided into 2 parts - LEFT and RIGHT memory. The exchange of columns between the processors take place along the arrows shown in Fig. 2. At the end of the frontend processing, all the sorted singular values and singular vectors reside in the memory of the first processor.

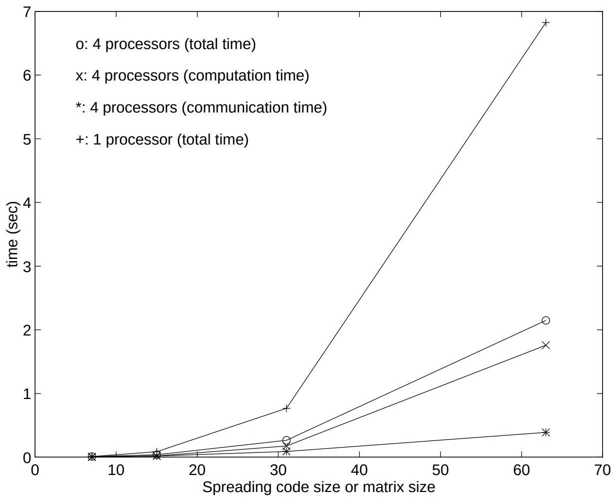
Performance measurements :

The performance of the implementation was tested using a random number generator to generate the transmitted data for each of the K users. Each user's spreading code was generated using length N (spreading code size) Gold codes. The use of the Gold codes as spreading codes restricts N to $2^k - 1$, k being an integer. So, while executing the frontend, N was varied from 7 to 63 and the blocksize L is equal to N. That is, a $N \times N$ matrix was decomposed.

Any parallel implementation incurs two main cost components - computation time, which includes execution time for all arithmetic/logic operations and communication time, which includes time for data movement between processors. The most commonly used criteria used to evaluate performance of a parallel algorithm is speedup, defined as $S_p = T_s/T_p$, where T_s is the time taken when the algorithm executes sequentially on one processor and T_p is the parallel run time. Ideally, the maximum speedup that can be obtained using p processors is p. However, due to factors like memory access conflicts and communication delays, the speedup in most real implementations is less than p.



(a)



(b)

Figure 3: Frontend - speedup and execution times vs. spreading code size (N)

In parallel processing, speedup figures are usually considered to be more important than absolute timings as speedup shows how much an application gains by parallelization. Hence, the speedup figures are reported (Fig. 3(a)) for the frontend as the input matrix size is increased (that is as the spreading code size is increased). The speedup increases as the size of the matrix increases. This is to be expected as the communication overhead decreases when compared to the computation time as the matrix size increases. As N increases, the speedup increases to 3.2, that is, a maximum processor utilization of $3.2/4 \times 100\% = 80\%$.

However, due to high real time requirements of the CDMA acquisition problem, there is also a need to measure and analyze the actual execution times of the implementation (Fig. 3(b)). The results show that the parallel implementation can handle 31 columns, each containing 31 elements in 0.26s, that is, each column is handled in 0.008s (0.26/31s). In this implementation, the data resides in the off-chip local memory. Code optimizations such as prefetching data into the on-chip RAM while DMA data transfer occurs between processors should considerably improve performance.

4.2. Implementation of deterministic MUSIC estimator on the PPDS

The use of the MUSIC estimator results in the optimization problem previously described by Eq. 5. For any integer $i \in [0, N)$, the minimization problem for $\tau \in [iT_c, (i+1)T_c]$ may be written as:

$$\hat{\tau}_k^{(i)} = T_c \times (i + \arg \min_{\delta \in [0,1]} ||((1-\delta)a_k(iT_c) + \delta a_k((i+1)T_c))' \mathcal{S}_N||_2^2) \tag{6}$$

If $a_k(iT_c)'\mathcal{S}_N$ is denoted by $b_{k,i}{}'$, then

$$\hat{\tau}_k^{(i)} = T_c \times (i + \arg\min_{\delta\in[0,1]} [(1-\delta)^2 b_{k,i}{}'b_{k,i} + 2\delta(1-\delta)b_{k,i}{}'b_{k,i+1} + \delta^2 b_{k,i+1}{}'b_{k,i+1}]) \quad (7)$$

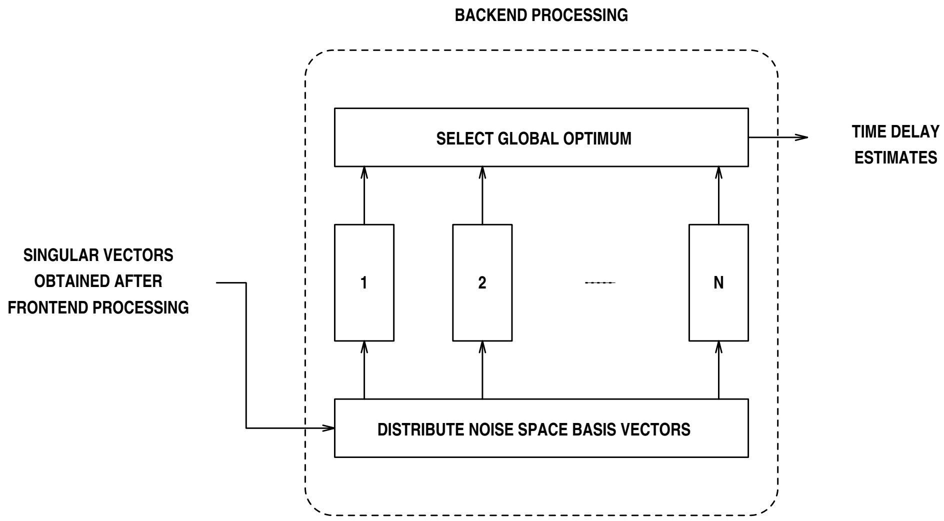


Figure 4: Parallel architecture to implement deterministic MUSIC for each single user

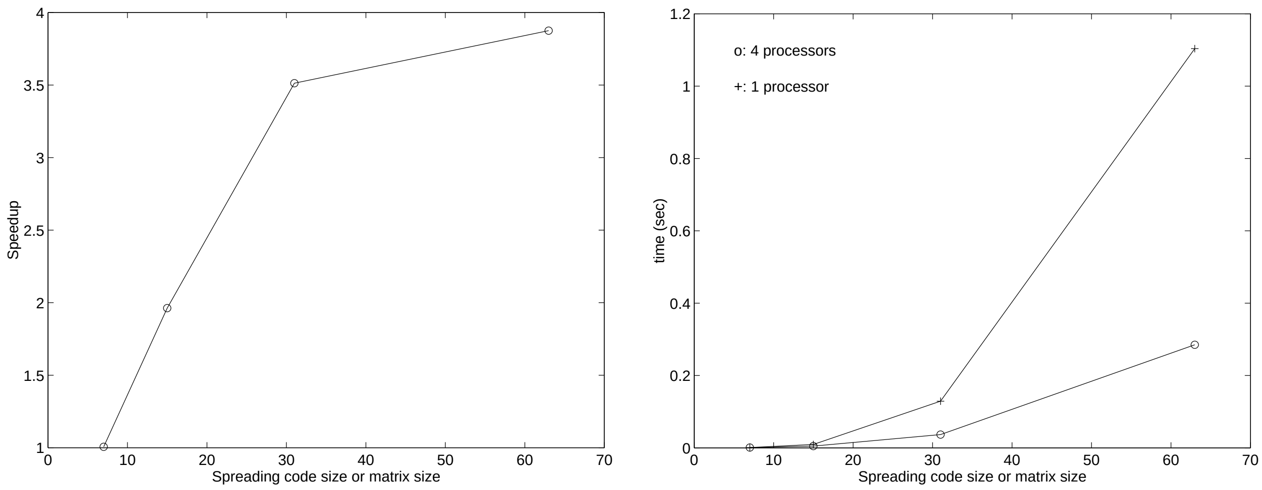
So, the backend computation using the deterministic MUSIC estimator consists of the following steps :

1. The matrix A_k is formed as follows:
$$A_k = [a_k(0) \;\; a_k(T_c) \;\; a_k(2T_c) \;\; \cdots \;\; a_k(NT_c - T_c)] \quad (8)$$
2. The matrix A_k is projected onto the basis for the noise subspace $\mathcal{S}_N$.
$$B_k = [b_{k,0} \;\; b_{k,1} \;\; b_{k,2} \;\; \cdots \;\; b_{k,N-1}]' = A_k'\mathcal{S}_N \quad (9)$$
3. $b_{k,i}{}'b_{k,i}$ and $b_{k,i}{}'b_{k,(i+1)modN}$ are computed for all i
4. $\hat{\tau}_k$ is computed using the $b_{k,i}{}'b_{k,i}$ and $b_{k,i}{}'b_{k,(i+1)modN}$

The formulation of the backend as a parallel algorithm using N processors is described in Fig. 4.[13] The backend was also implemented on the PPDS with the 4 processors configured as a linear array. The processing corresponding to the columns of the parallel array shown in Fig. 4 were distributed among the 4 processors of the PPDS. The architecture consists mainly of a linear array for the matrix multiplication $A_k'\mathcal{S}_N$ (step 2, above). The same array is also used to perform the dot product and calculation of delays(τ) specified in steps 3-4.

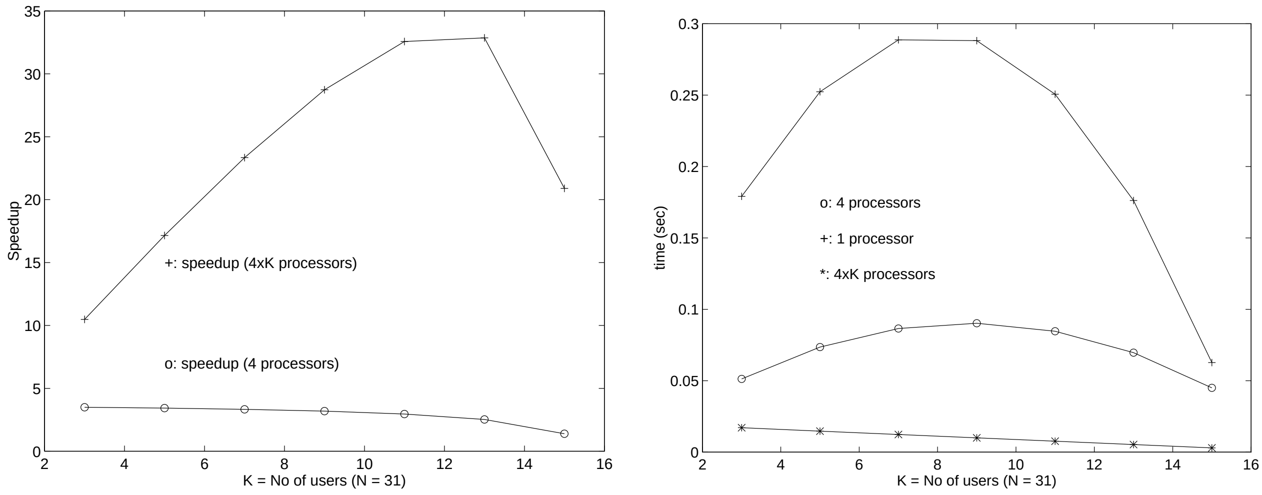
Performance measurements :

Two separate experiments were carried out in this section. In the first, (Fig. 5) the dependence of speedup and execution time of the backend on the spreading code size was analyzed with the number of users being fixed at 2. In the second, (Fig. 6) speedup and execution times were measured as the number of users, K was varied while the spreading code size was fixed at $N = 31$.



(a) (b)

Figure 5: Backend - speedup and execution times vs. spreading code size (N)



(a) (b)

Figure 6: Backend - speedup and execution times vs. number of users (K)

In the backend, the work for each user, was divided among the 4 processors. However, since the estimation of the delay of each user is independent of the delays of all other users, an obvious approach towards parallelization would be to perform the delay estimate of each user on separate sets of 4 processors. These experiments were done on one PPDS, with only 4 processors, that is, the work for each user was divided among the 4 processors, but the K users were handled sequentially. However, the execution times for estimating delays of all K users on 4 processors can easily be extrapolated to execution times for estimating delays of all K users on $4 \times K$ processors. That is, the delay of each user is estimated on separate sets of 4 processors. This extrapolation is done using the relation : $t_{4\times K} = t_4/4$, where t_4 is the execution time of delays of K users on 4 processors and $t_{4\times K}$ is the the execution time of delays of K users on $4 \times K$ processors. Fig. 6(b) shows the variation of both t_4 and $t_{4\times K}$ with K.

An important point to be noted from Fig. 6(b), is that the execution time for the backend, on 1 or 4 processors, increases as K increases, until $K = N - 2K$. As K increases further the execution time for the backend decreases. This is because, on the one hand, the size of the noise subspace $\mathcal{S}_N$, ($\mathcal{S}_N$ comprises the $N - 2K$ smallest singular vectors), decreases as K increases. So, for each user, there is less work to do in the estimation steps, as K increases. On the other hand, since the 4 processors handle the K users sequentially, there are more delays to be estimated as K increases. However, with $4 \times K$ processors, the K users are handled in parallel, so the execution time continuously decreases as K is increased.

Lastly, it may be noted that the execution time for the backend using the deterministic MUSIC estimators is much less than that for the frontend using 4 processors. As explained in the previous paragraph, the backend can be improved further using $4 \times K$ processors. So, the effort in subsequent stages of this research needs to be concentrated on improving the performance of the frontend, by trying out the different approaches for computing the SVD, discussed in section 3.1.

5. CONCLUSIONS

Various algorithms for solving both the frontend and the backend processing in the subspace-based channel estimation system have been analyzed. Since the aim of this research was to get a working prototype of a parallel system implementing both the frontend and the backend, appropriate algorithms best suited for implementation on the TMS320C40 Parallel Processing Development System was chosen. The performance analysis of the implementation shows good processor utilization due to speedups close to 3.8 using 4 processors and an appropriate data size. This shows that the chosen parallelization schemes are suitable for the TMS320C40 architecture. Also, the execution time of the frontend is much more than that of the backend which implies that the effort in the next phase needs to be concentrated on improving the performance of the frontend.

6. ACKNOWLEDGMENTS

This work was supported in part by Nokia Corporation, by the Texas Advanced Technology Program under grant #003604-049, and by NSF under grant NCR 9506681.

7. REFERENCES

[1] S. E. Bensley and B. Aazhang. Subspace-based channel estimation for code divisio n multiple access communication systems. *Accepted for publication in IEEE Trans. Commun*, 1996.

[2] R. O. Schmidt. *A signal subspace approach to multiple emitter location and spectral estimation.* PhD thesis, Stanford Univ., Stanford, CA, 1981.

[3] G. H. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *SIAM J. Num. Anal.*, B(2):205–224, 1965.

[4] R. P. Brent, F. T. Luk, and C. F. Van Loan. Computation of the Singular Value Decomposition Using Mesh-Connected Processors. *Journal of VLSI and Computer Systems*, 1(3):242–270, 1985.

[5] R. Schreiber. Solving eigenvalue and singular value problems on an undersized systolic array. *SIAM Journal Sci. Stat. Comput.*, 7(2):441–451, April 1986.

[6] P. J. Eberlein and H. Park. Efficient implementation of Jacobi algorithms and Jacobi sets on distributed memory architectures. *Journal of Parallel and Distributed Computing*, 8(4):358–366, April 1990.

[7] K. Kota and J. R. Cavallaro. Pipelining multiple SVDs on a single processor array. In Franklin T. Luk, editor, *SPIE, Advanced Signal Processing Algorithms, Architectures, and Implementations V*, volume 2296, pages 612–623, 1994.

[8] J. R. Bunch and C. P. Neilsen. Updating the singular value decomposition. *Numer. Math.*, 31:111–129, 1978.

[9] M. Moonen, P. van Dooren, and J. Vandewalle. A singular value decomposition updating algorithm for subspace tracking. *SIAM J. Matrix Anal. and Appl.*, 13(4):1015–1038, October 1992.

[10] Texas Instruments. *Application Guide, Parallel processing with the TMS320C4x*, 1994.

[11] Texas Instruments. *Technical Reference, TMS320C4x Parallel Processing Development System*, 1993.

[12] B. A. Curtis and V. K. Madisetti. Rapid prototyping on the Georgia Tech digital signal multiprocessor. *IEEE Transactions on Signal Processing*, 42(3):649–661, March 1994.

[13] K. Kota and J. R. Cavallaro. Parallel algorithms for CDMA synchronization. *ECE Department, Rice University, Technical Report*, (9409), November 1995.