VLSI Implementation of Conjugate Gradient Based Mobile User Tracking System

Ramin Baghaie and Petri Karttunen
Laboratory of Telecommunications Technology
Helsinki University of Technology
P.O. BOX 3000, FIN-02015 HUT, Finland
e-mail: {ramin.baghaie, petri.karttunen}@hut.fi

***Abstract*- This paper considers the implementation of a signal subspace based mobile user tracking system that utilizes an efficient Conjugate Gradient (CG) based step-by-step adaptation scheme. First, we estimate the computational complexity of different units of the tracking system. Based on these estimations, we partition the implementation task into two parts: software and hardware. Finally, for the hardware implementation of the tracking unit a systolic architecture is proposed. With the aid of the proposed systolic array the time complexity of the tracking unit is reduced to *O*(*M*).**

I. Introduction

The channel parameter tracking problem arises in numerous situations. One example is mobile user tracking in which the spatial beamforming procedure must be carried out continuously for each user. In [1], for the mobile user tracking system, a step-by-step update scheme of the CG method was implemented. It was shown that the proposed CG based tracking system has a better tracking performance in terms of faster and smoother convergence and smaller misadjustment.

In this paper, we consider the VLSI implementation of the tracking system of [1], and focus on developing efficient systolic architectures that are suitable for real-time applications. This paper is organized as follows. Section II briefly presents the structure of the CG based tracking system. In Section III, the computational complexity of the system is evaluated. In Section IV, implementation issues of the tracking system are discussed. Furthermore, for the tracking unit a novel systolic architecture is designed that reduces the required computational time by an order of magnitude. Finally, concluding remarks are provided in Section V.

II. System Model

Figure 1 illustrates the overall system model of our tracking system [1]. In this section, we briefly describe the function of each unit.

This work is part of a research project of the Institute of Radio Communication (IRC) funded by the National Technology Agency (TEKES), NOKIA Research Center, Sonera Ltd., and the Helsinki Telephone Company.

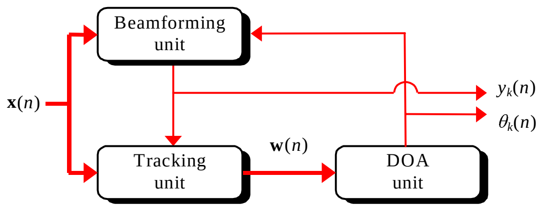


Fig. 1. The overall system for the signal subspace based user tracking [1]

A. Tracking Unit

The signal subspace tracking problem is formulated as the quadratic cost function for which a step-by-step update scheme has been implemented [1].

For the step-by-step update scheme the modified CG (MCG) algorithm has been utilized [2]. In the MCG algorithm of Table I, $\alpha(n)$ is the step size that minimizes the cost function through the line search procedure along the search direction $\mathbf{p}_{n-1}$. The residual vector $\mathbf{g}(n)$ points to the direction of the steepest descent. λ_f is the forgetting factor and η should be $(\lambda_f - 0.5) \le \eta \le \lambda_f$ [2]. Factor $\beta(n)$ ensures that the *R-orthogonality* is preserved between search directions.

TABLE I
Sample-by-Sample CG (MCG) Algorithm

Set initial conditions: $\mathbf{w}(0) = \mathbf{0}$, $\mathbf{g}(0) = \mathbf{b}(0)$, $\mathbf{p}(0) = \mathbf{g}(0)$

for n = 1, 2, …

$$\alpha(n) = \eta \frac{\mathbf{p}^{\mathrm{H}}(n-1)\mathbf{g}(n-1)}{\mathbf{p}^{\mathrm{H}}(n-1)\mathbf{R}(n-1)\mathbf{p}(n-1)}$$

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \alpha(n)\mathbf{p}(n-1)$$

$$\mathbf{g}(n) = \lambda_f \mathbf{g}(n-1) - \alpha(n)\mathbf{R}(n-1)\mathbf{p}(n-1) + \mathbf{x}(n)(d(n) - \mathbf{x}^{\mathrm{H}}(n)\mathbf{w}(n-1))$$

$$\beta(n) = \max\left\{ \frac{(\mathbf{g}(n) - \mathbf{g}(n-1))^{\mathrm{H}}\mathbf{g}(n)}{\mathbf{g}^{\mathrm{H}}(n-1)\mathbf{g}(n-1)}, 0 \right\}$$

$$\mathbf{p}(n) = \mathbf{g}(n) + \beta(n)\mathbf{p}(n\text{-}1)$$

end

In our system derivation, for the sake of simplicity initially unknown antenna response vector $\mathbf{a}(\theta_k)$ has been replaced with a weight vector estimate $\mathbf{w}(n)$. The weight vector $\mathbf{w}(n)$ as provided by the tracking unit converges to the desired steering vector which correlates best the desired user signal.

B. DOA Extraction Unit

In the Direction-of-Arrival (DOA) extraction unit, the new tracking angle estimates $\theta_k(n)$, (k=1,…,N) are computed through the Least Square (LS) fitting criterion that are based on the small deviations in the array manifold [1].

The LS criterion is based on the linear model and can be expressed as:

$$\hat{\theta}_k^{(\mathrm{LS})}(n) = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H}\right)^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{z}_k(n) \tag{1}$$

where $\mathbf{z}_k(n)$ consists of array samples of θ_k, and $\mathbf{H}$ is the $M\times 1$ observation vector.

C. Beamforming Unit

In this unit, the following conventional beamforming method has been utilized [1].

$$y_k(n) = \left(\mathbf{w}^{\mathrm{H}}(n)\mathbf{w}(n)\right)^{-1}\mathbf{w}^{\mathrm{H}}(n)\mathbf{x}(n) \tag{2}$$

III. Complexity of the System

In this section, the computational complexity of different units are estimated and compared. In the tracking, DOA, and beamforming units, the most computationally intensive operations are the calculation of step size α, Eq. (1), and Eq. (2), respectively. In Table II for N sources, the order of computational complexity for different units of the tracking system is calculated.

TABLE II
COMPARISON OF COMPUTATIONAL COMPLEXITIES

Unit	*Order of complexity for N sources*
Tracking	$O(NM^2)$
DOA	$O(NM)$
Beamforming	$O(NM)$

M: Number of antennas
N: Number of sources

As can be seen from Table II, the tracking unit has the highest order of complexity. The core of this unit is the sample-by-sample CG algorithm. As compared to the conventional CG algorithm also referred to as Block Conjugate Gradient (BCG), in the MCG algorithm, the computation of the residual vector $\mathbf{g}(n)$ and the factor $\beta(n)$ are more complex and require a higher number of vector inner products. Next, we study and compare the computational complexities of the sample-by-sample CG and the BCG algorithms. The results are shown in Table III.

It is clear that the computational complexity of the BCG depends on the number of iterations I and for a large M, the BCG is I times more complex than the sample-by-sample CG algorithm.

TABLE III
COMPARISON OF COMPUTATIONAL COMPLEXITIES OF TWO CG ALGORITHMS

Algorithm	*Number of complex multiplications*
BCG	$I(M^2+5M+2)-2M-1$
MCG	$M^2+10M+3$

I: Maximum number of iterations for a block
M: Number of antennas

IV. Implementation Of The System

For real-time applications, in order to meet the demand of high sampling rates the conventional DSP-based implementation methods are not sufficient. Consequently, for the implementation of units with high computational complexity, application-specific integrated circuits (ASIC) should be utilized.

As can be seen from Table II, in our system the most computationally intensive block is the tracking unit. In this unit, the order of complexity for N sources is $O(NM^2)$. In Figure 2, the hardware (HW)/software (SW) partitioning of our system is illustrated.

In this section, we discuss the implementation of the HW partition that is needed for the MCG algorithm and focus on developing an efficient VLSI array processor that is suitable for real time applications. For this purpose, we design a systolic array that targets the most computationally intensive block of the MCG algorithm.

A. Review of the Implementation Techniques

As can be seen from Table I, in our tracking unit the most computationally intensive operations are the matrix computations. Furthermore, in order to meet the demand for high sampling rates and to achieve acceptable execution speed the conventional serial implementation methods are not sufficient. Thus, parallel architectures should be utilized.

For the matrix-vector computations needed in the MCG algorithm of Table I, several classes of parallel architectures such as multiprocessors, systolic-type arrays, vector computers and array computers have been proposed [3].

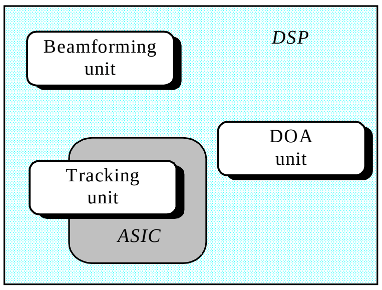


Fig. 2. Partitioning of the overall system into HW/SW

Although many of these parallel architectures have demonstrated their effectiveness for executing matrix-vector computations, due to their broadcasting or complex interconnection network they may not be suitable for VLSI implementation. These drawbacks led to the introduction of application-specific architectures and in particular systolic arrays, which are natural for matrix operations. They match the fine granularity of parallelism available in the computations and have very low overhead in communication and synchronization. In addition, the regular nature of systolic-type arrays meets the requirements for effective use of VLSI [3,4].

Although, for the matrix operations needed in the MCG algorithm, a variety of systolic architectures exists, the main problem is to map the entire algorithm onto a suitable and practical VLSI architecture. In the MCG however, due to the serial nature of the algorithm, there is a very low degree of parallelism and therefore, parallelization of the algorithm is not trivial. Similarly, this is the case when implementation of the Block conjugate Gradient algorithm is of interest [5],[6].

B. Systolic implementation

In this section, we design a systolic architecture that reduces the time complexity of the MCG algorithm to $O(M)$. As discussed in the previous section, due to the serial nature of the algorithm, there is a very low degree of parallelism in the algorithm. Furthermore, due to the iterative nature of the algorithm and the requirement for different resetting schemes for β [2], direct mapping of the MCG algorithm to ASIC is not practical. Therefore, our systolic architecture targets the matrix-vector and vector-vector products needed in the calculation of the step size α and the factor β.

Consider the calculation of the step size α:

$$\alpha(n) = \eta \frac{\mathbf{p}^{\mathrm{H}}(n-1)\mathbf{g}(n-1)}{\mathbf{p}^{\mathrm{H}}(n-1)\mathbf{R}(n-1)\mathbf{p}(n-1)} \tag{3}$$

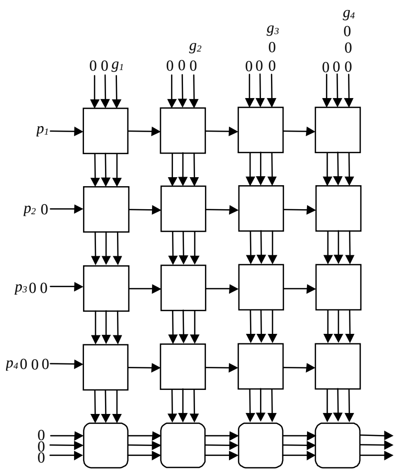
For simplicity, we introduce the new variable $\mathbf{v}(n)$ as follows:

$$\mathbf{v}(n) = \mathbf{R}(n)\mathbf{p}(n) \tag{4}$$

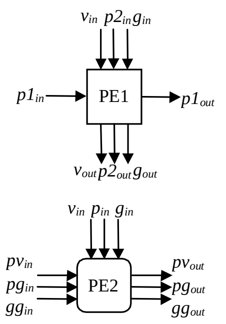
Due to the sample-by-sample update scheme in the MCG algorithm, the correlation matrix $\mathbf{R}(n)$ varies in every sample. However, when calculating the weight vectors for N individual sources, $\mathbf{R}(n)$ remains the same and therefore, for N iterations the same $\mathbf{R}(n)$ is used. As a result, for the systolic architecture a 2D array implementation is adopted. The elements of the $\mathbf{R}(n)= r_{ij}(n)$ $(i,j = 1, \ldots, M)$ are preloaded into this array processor and remain constant for N iterations. Now, consider the following vector-vector multiplications that are needed in the MCG algorithm.

$$\mathbf{pg}(n-1) = \mathbf{p}^{\mathrm{H}}(n-1)\mathbf{g}(n-1) \tag{5}$$

$$\mathbf{pv}(n) = \mathbf{p}^{\mathrm{H}}(n-1)\mathbf{v}(n) \tag{6}$$



(a)



v_{in} $p2_{in}$ g_{in}
$p1_{in}$
PE1
$p1_{out}$
v_{out} $p2_{out}$ g_{out}
v_{in} p_{in} g_{in}
pv_{in}
pg_{in}
gg_{in}
PE2
pv_{out}
pg_{out}
gg_{out}

$$v_{out} \leftarrow v_{in} + p1_{in} \cdot r_{ij}$$
$$g_{out} \leftarrow g_{in}$$
$$p1_{out} \leftarrow p1_{in}$$
$$p2_{out} \leftarrow p2_{in} \quad if\ i \neq j$$
$$p2_{out} \leftarrow p1_{in} \quad if\ i = j$$
$$pv_{out} \leftarrow pv_{in} + p_{in}^{*} \cdot v_{in}$$
$$pg_{out} \leftarrow pg_{in} + p_{in}^{*} \cdot g_{in}$$
$$gg_{out} \leftarrow gg_{in} + g_{in}^{*} \cdot g_{in}$$

(b)

Fig. 3. a) Systolic architecture when M=4
b) Input-output ports and cell functions

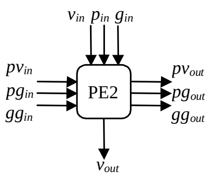
For the realization of Eq. (5) and Eq. (6), a linear array is selected. For synchronization purposes, the linear array is placed below the 2D array.

Figure 3 illustrates the proposed systolic array when M=4. In Figure 3b, the cell function of each Processor Element (PE) is illustrated.

Furthermore, this architecture utilizes the availability of the residual vector $\mathbf{g}(n)$ and performs the following vector inner product needed in the calculation of β.

$$\mathbf{gg}(n-1) = \mathbf{g}^{\mathrm{H}}(n-1)\mathbf{g}(n-1) \tag{7}$$

As can be seen from Table I, for the calculation of the residual vector $\mathbf{g}(n)$, matrix-vector multiplication of (4), i.e. vector $\mathbf{v}(n)$, is required. For utilizing $\mathbf{v}(n)$ two methods can be exercised. One method is to keep the elements of $\mathbf{v}(n)$ by allocating a local memory to each PE2 and then sequentially transfer them to the host from the last PE2 of the linear array. The second method is to slightly modify the PE2s. This can be achieved by adding an extra output port to the PE2s as it is illustrated in Fig. 4. The total number of PEs required in this systolic architecture is M^2+M.



$$pv_{out} \leftarrow pv_{in} + p_{in}^{*} \cdot v_{in}$$
$$pg_{out} \leftarrow pg_{in} + p_{in}^{*} \cdot g_{in}$$
$$gg_{out} \leftarrow gg_{in} + g_{in}^{*} \cdot g_{in}$$
$$v_{out} \leftarrow v_{in}$$

Fig. 4. I/O ports and cell functions of the modified PE2

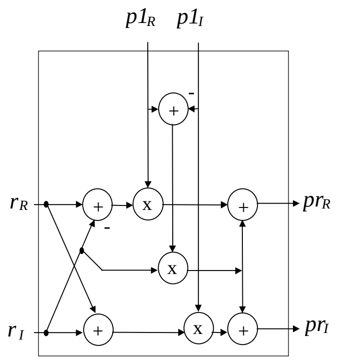


Fig. 5. Implementation of a complex multiplier by using the SR technique

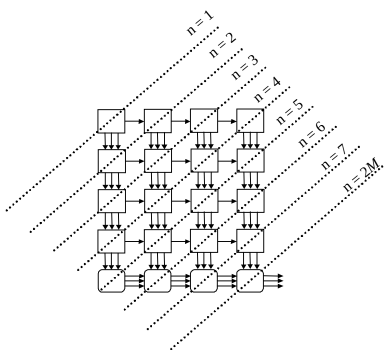


Fig. 6. Illustration of the data movement for each computation step

For the implementation of the complex multipliers needed in the PEs, Strength Reduction (SR) transformation technique has been utilized [7]. By utilizing the SR transformation the total number of real multiplications needed in a complex multiplier is reduced to only three. To clarify this further, consider the complex multiplications required in PE1, i.e. $pr = (pr_R + jpr_I) = p1 \cdot r$. By utilizing the SR technique we have:

$$pr_R = (p1_R - p1_I)r_I + p1_R(r_R - r_I) \tag{8a}$$

$$pr_I = (p1_R - p1_I)r_I + p1_I(r_R + r_I) \tag{8b}$$

As can be seen from Eq. (8) and Figure 5, by utilizing the SR transformation the total number of real multiplications needed in a complex multiplication is reduced to only three. This is at the expense of having three additional adders. However, it is well known that multiplications are more complex than additions and consume much more power as well. In fact, for a single complex multiplication power reductions of up to 25% can be achieved [7]. Thus, the SR transformation can result in remarkable savings in consumed power and silicon area.

In order to calculate the throughput of the systolic array, we assume that one time step of the global clock corresponds to the processing time required for each PE. For the initialization of PE1s, *M* time steps are needed. Thus, the total computation time required by the array is 3*M* steps. Figure 6 illustrates the flow of data in the proposed systolic architecture for different time steps.

V. Conclusions

In this paper, implementation of a signal subspace based mobile user tracking system was discussed that utilizes an efficient sample-by-sample CG algorithm. First, the computational complexity of different units of our mobile user tracking system was estimated. Based on these estimations, for a more realistic implementation, the tracking system was partitioned into two parts: HW and SW. For the hardware implementation of the tracking unit, a systolic architecture was proposed. With the aid of this systolic array, the time complexity of the most computationally intensive unit was reduced to $O(M)$. Furthermore, for the implementation of the complex multipliers needed in the PEs, Strength Reduction transformation technique was utilized. As a result, remarkable savings in consumed power and silicon area were achieved. Future research should be directed towards mapping the system into a fixed number of processors when the number of antennas *M* is large.

References

[1] P. Karttunen and R. Baghaie, "Conjugate Gradient Based Signal Subspace Mobile User Tracking," in *Proceedings IEEE Vehicular Technology Conference*, Houston, USA, vol. 2, pp. 1172-1176, May 1999.

[2] S.P. Chang and A.W. Willson, "Adaptive filtering using modified conjugate gradient," in *Proceedings 38th Midwest Symposium on Circuits and Systems*, Rio de Janeiro, Brazil, pp. 243-246, August 1995.

[3] J.H. Moreno and T. Lang, "Matrix computations on systolic-type meshes," *IEEE Computer*, pp. 32-51, April 1993.

[4] S.Y. Kung, *VLSI Array Processors.* Englewood Cliffs, New Jersey: Prentice Hall, 1988.

[5] J. Tasic, M. Gusev, and D.J. Evans, "Systolic implementation of preconditioned conjugate gradient method in adaptive transversal filters," *Parallel Computing*, vol. 18, no. 9, pp. 1053-1065, Sept. 1992.

[6] Y. Saad, "Practical use of polynomial preconditionings for the conjugate gradient method," *SIAM Journal of Scientific Statistical computing*, vol. 6, no. 4, pp. 865-881, October 1985.

[7] N. Shanbhag and M. Goel, "Low-Power adaptive filter architectures and their applications to 51.84 Mb/s ATM-LAN," *IEEE Transactions on Signal Processing*, vol. 45, no. 5, pp. 1276-1290, May 1997.