

An Efficient Transputer Implementation of a Systolic Architecture for Parallel Kalman Filtering

A. Seddiki, N. Taleb, and D. Guerchi

Abstract—The computations involved in Kalman filter algorithms are highly structured with regular matrix-type operations. Moreover, Kalman filter algorithms can be specified as a set of parallel passes rearranged to be the type of the Fadeev algorithms for generalizing matrix/vector manipulations. Based on that, and in order to further improve the speed of updating the state estimate of Kalman filter equations and taking full advantage of the two parallel pipeline structures, a new combined SIMD/MISD (Single Instructions Multiple Data stream/ Multiple Instructions Single Data stream) Transputer implementation scheme is presented. This configuration has expanded the utilization of the two-dimensional systolic architecture and significantly improved the speed of updating the state estimate and resulted in a significant saving in processing time complexity compared to SIMD transputer implementation.

Index Terms—Kalman filter, fadeev algorithm, transputers, occam language, systolic arrays.

I. INTRODUCTION

Kalman filters are a class of linear minimum variance algorithms for the estimation of the state vectors from measurement data corrupted by noise. A minimum variance estimate has the property that its error variance is less than or equal to that of any other estimate. The applicability of the Kalman filter to real-time processing problems is generally limited, however, by the relative complexity of the matrix computation involved. The number of arithmetic operations required for implementing the algorithm with n state variables is $O(n^2)$ and $O(n^3)$ for the covariance updates. These require high-speed, real time computations which must be performed on continuous data streams. The computations involved are highly structured with regular matrix-type operations, a feature shared Kalman filter algorithms. One approach for parallel implementation attempts to decompose the algorithm into a set of basic calculations through a fundamental reorganization of the computing architecture, of which systolic arrays are an example. The choice of arrays of transputers may then be made on the basis of factors such as size, cost, and computational bandwidth required. The aim of the work reported in this paper is to investigate an efficient way to speed-up the updating of the state estimate of the parallel

Kalman filtering method using transputers and their language Occam [1]-[4]. The design of parallel Kalman algorithm is presented in the Section II, followed by the computing modes for both internal and the boundary cells of systolic arrays of filtering passes in Section III. Occam simulation based transputer implementation is discussed in Section IV. Finally, the processing time complexity of the new SIMD/MISD configuration compared to the SIMD one is performed in Section V.

II. PARALLEL KALMAN CONCEPT

The application of the Fadeev algorithm to Kalman filtering is introduced in this section [5].

Consider the linear, discrete-time system described by,

$$x(k+1) = F(k)x(k) + w(k)$$

$$z(k) = H(k)x(k) + v(k)$$

where $x(k)$ is the $(n \times 1)$ state vector and $z(k)$ is the $(m \times 1)$ measurement vector. The state transition matrix $F(k)$ and measurement matrix $H(k)$ are of appropriate dimensions. The system and the measurement noise sequences $w(k)$ and $v(k)$ are assumed to be independent, zero mean Gaussian white noise sequences with covariance matrices $Q(k)$ and $R(k)$ respectively.

The Kalman filter algorithm for optimal state estimation takes the following form.

Time updates:

$$\hat{x}(k/k-1) = F(k-1) \hat{x}(k-1/k-1) \quad (1)$$

$$P(k/k-1) = F(k-1)P(k-1/k-1)F^T(k-1) + Q(k-1) \quad (2)$$

Measurement updates:

$$P^{-1}(k/k) = P^{-1}(k/k-1) + H^T(k)R^{-1}(k)H(k) \quad (3)$$

$$K(k) = P(k/k)H^T(k)R^{-1}(k) \quad (4)$$

$$\tilde{z}(k) = z(k) - H(k) \hat{x}(k/k-1) \quad (5)$$

$$\hat{x}(k/k) = \hat{x}(k/k-1) + K(k) \tilde{z}(k) \quad (6)$$

The initial conditions are

$$\hat{x}(0/0) = 0, P(0/0) = P_0 \text{ or } P^{-1}(0/0) = P_0^{-1}, k=1,2,\dots$$

Manuscript received July 1, 2013; revised September 13, 2013.

A. Seddiki is with Laboratory of Telecommunications & Digital Signal Processing, University of Sidi-bel-Abbes, Algeria (e-mail : Seddiki_ali@msn.com)

N. Taleb is with Laboratory of Communication Networks, Architecture, and Multimedia, University of Sidi-bel-Abbes, Algeria.

D. Guerchi is with Canadian University of Dubai, Computer Vision and Speech Processing Center, United Arab Emirates.

The vector $\hat{x}(k/k)$ represents the optimal estimate of $x(k)$ based on the measurement sequence $\{z(1), z(2), \dots, z(k)\}$.

The fadeev algorithm can be introduced in terms of calculating:

$$WB + D \quad (7)$$

Given

$$WA = C, \quad (8)$$

where A, B, C, D are known matrices and W is unknown. Clearly (8) gives W as:

$$W = CA^{-1}$$

This can be substituted in (7) to produce the required result:

$$CA^{-1}B + D$$

The same answer can be arrived at, without the necessity of finding W explicitly, as follows. Matrices A, B, C and D are loaded into compound matrix:

$$\begin{bmatrix} A & B \\ -C & D \end{bmatrix} \quad (9)$$

The Fadeev algorithm then involves reducing C to zero,

in the compound matrix of (9), by ordinary row manipulation of the Gaussian elimination type. In practice a modified Fadeev algorithm, where matrix A is changed to triangular form prior to annulment of C , is more suitable for systolic array processing [5], [6]. This is summarized as,

$$\begin{bmatrix} A & B \\ -C & D \end{bmatrix} \rightarrow \begin{bmatrix} TA & TB \\ -C & D \end{bmatrix} \rightarrow \begin{bmatrix} TA & TB \\ -C & D + CA^{-1}B \end{bmatrix} \quad (10)$$

where TA is an upper triangular matrix.

In a particular way, it is showed that matrix equations (1) to (6) of the Kalman filter algorithm could be produced in eight successive passes of the Fadeev algorithm as shown in Table I.

TABLE I: APPLICATION OF FADEEV ALGORITHM TO KALMAN FILTER EQUATIONS

Pass	A	B	C	D	Result
1	I	$\hat{x}(k-1/k-1)$	$-F(k-1)$	0	$\hat{x}(k/k-1)$
2	$P^{-1}(k-1/k-1)$	$F^T(k-1)$	$-F(k-1)$	$Q(k-1)$	$P(k/k-1)$
3	$R(k)$	I	$-H^T(k)$	0	$H^T(k)R^{-1}(k)$
4	$P(k/k-1)$	I	$-I$	0	$P^{-1}(k/k-1)$
5	I	$H(k)$	$-H^T(k)R^{-1}(k)$	$P^{-1}(k/k-1)$	$P^{-1}(k/k)$
6	$P^{-1}(k/k)$	$-H^T(k)R^{-1}(k)$	I	0	$K(k)$
7	I	$\hat{x}(k/k-1)$	$H(k)$	$z(k)$	$\hat{z}(k)$
8	I	$\hat{z}(k)$	$-K(k)$	$\hat{x}(k/k-1)$	$\hat{x}(k/k)$

0	0	0	...	0	0	D_{nn}
..	$D_{(n-1)n}$
..
..	0	D_{41}	D_{32}	D_{23}	...	B_{5n}
0	0	0	...	$-C_{4n}$	D_{31}	D_{22}	D_{13}	...	B_{4n}
0	0	$-C_{n3}$...	$-C_{3n}$	D_{21}	D_{12}	B_{n3}	...	B_{3n}
0	$-C_{n2}$	$-C_{(n-1)3}$...	$-C_{2n}$	D_{11}	B_{n2}	$B_{(n-1)3}$...	B_{2n}
$-C_{n1}$	$-C_{(n-1)2}$	$-C_{(n-2)3}$...	$-C_{1n}$	B_{n1}	$B_{(n-1)2}$	$B_{(n-2)3}$...	B_{1n}
..
$-C_{21}$	$-C_{12}$	A_{n3}	...	A_{3n}	B_{21}	B_{12}	0	...	0
$-C_{11}$	A_{n2}	$A_{(n-1)3}$...	A_{2n}	B_{11}	0	0	...	0
A_{n1}	$A_{(n-1)2}$	$A_{(n-2)3}$...	A_{1n}	0	0	0	...	0
..
A_{31}	A_{22}	A_{13}	...	0	0
A_{21}	A_{12}	0	...	0	0
A_{11}	0	0	...	0	0

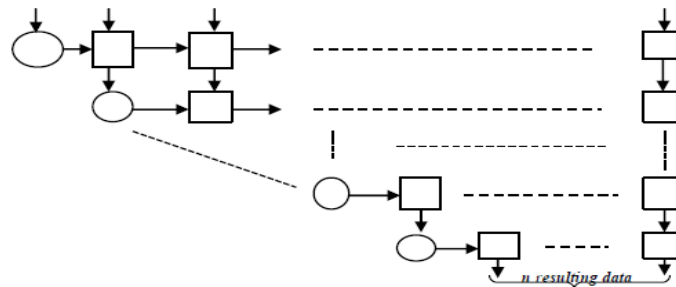


Fig. 1. Data flow and cell arrangement for one filtering pass.

III. SYSTOLIC ARCHITECTURE

The architecture for performing the computation outlined

in the last section is based on LU decomposition. Fig. 1 contains a schematic diagram of the array for n state variables, where each filtering pass is a two-step operation

involving triangularization of matrix A followed by transforming matrix C into a null matrix (10). This was achieved in by defining separate computing modes for both the internal and the boundary cells. Thus the data for A and B would be fed through the array first in the triangularization mode. With this step completed, the array would then accept the elements of C and D matrices and nullify C , using diagonal elements of upper triangular matrix A stored in local memory as pivots [5]-[7].

The cell operations are defined as follows,

Boundary cell: Fig. 2(a)

- Mode 1 for triangularization of A :

If $X_{in} = 0$, $m = s = 0$

If $X_{in} > mem$, $m = mem / X_{in}$, $s = 1$, $mem = X_{in}$

If $X_{in} < mem$, $m = X_{in} / mem$, $s = 0$, $mem = mem$

- Mode 2 for nullify of C :

If $mem = 0$, $m = s = 0$, $mem = mem$

If $mem \neq 0$, $m = X_{in} / mem$, $s = 0$, $mem = mem$

Internal cell: Fig. 2(b)

If $s = 1$, $X_{out} = mem - m * X_{in}$, $mem = X_{in}$

If $s = 0$, $X_{out} = X_{in} - m * mem$, $mem = mem$

The boundary cell finds the pivot between nearest neighbors and calculates the multiplying factor m .

The internal cell handles the rest of calculation involved in multiplying the elements of one row of a matrix by m and subtracting them from corresponding elements in succeeding rows, as needed for elimination. The variable

mem is data stored in local memory of a cell and s represents a control variable.

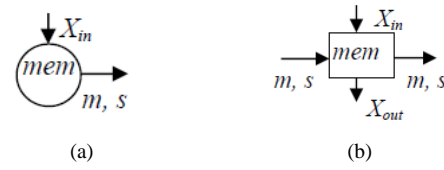


Fig. 2. Boundary and internal cells operations.

IV. TRANSPUTER IMPLEMENTATION

Simulation of parallel Kalman filtering has been simulated using Occam language. To perform the Kalman filtering process using Fadeev algorithm and cell arrangement achieved in Fig. 1, a detailed diagram built with all the channels named and numbered for different cells is shown in Fig. 3. Systolic array cells have no local memory, but in Occam simulation the cells only perform operations and cannot store data from one time-step to the next. Therefore, there has to be a global memory handler. At each time- step the data in cell memory is passed to the memory handler which returns it to the same cell at the beginning of the next time-step. Fig. 4 describes a computing flow diagram showing a parallel pipeline structure to establish the state estimate of Kalman filtering process executed in parallel manner and showing a parallel treatment of adequate passes of the fadeev algorithm applied to Kalman filter equations [5]-[7].

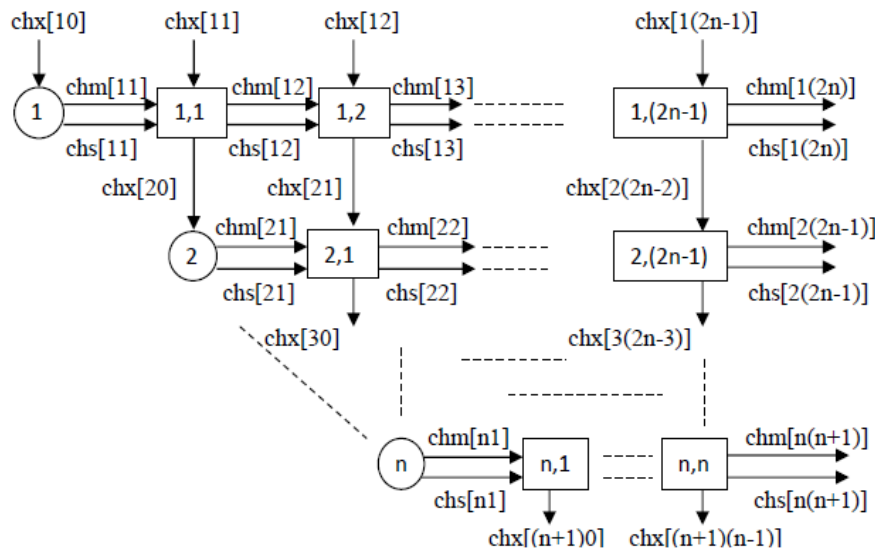


Fig. 3. Occam simulation of the array.

A configuration was developed showing a direct and suitable SIMD (*Single Instructions Multiple Data stream*) transputer implementation for this parallel pipeline structure (Fig. 5). In this configuration, a supervising transputer controls data input and output as well as results and two transputers treat simultaneously adequate parallel passes (same processes). The communication between transputers is accomplished through links corresponding to channels in

Occam programming [8]-[10].

According to the flow diagram (Fig. 4) and during each filtering pass at appropriate computing times, the output of Transputer T_1 is sent to Transputer T_2 . The Occam code for this implementation is the following:

```
{ { {
--Define channel addresses:
```

```

DEF Link.0.out=0;
DEF Link.0.in= 4;
DEF Link.1.in=5;
DEF Link.2.in=6;
  CHAN COM1, COM2, COM3:
--sc. Display.occ (sc. is a separately compiled procedure)
--sc. Main1.occ and sc. Main2.occ
  PLACED PAR
    PROCESSOR 0 T8 --Transputer T0
      PLACE COM1 AT Link.1.in:
      PLACE COM2 AT Link.2.in:
      Display (COM1, COM2, 0)
    PROCESSOR 1 T8 --Transputer T1
      PLACE COM1 AT Link.0.in:
      PLACE COM3 AT Link.2.out:
      Main1 (COM1, COM3, 1)
    PROCESSOR 2 T8 --Transputer T2
      PLACE COM2 AT Link.0.out:
      PLACE COM3 AT Link.2.in:
      Main2 (COM2, COM3, 2)
  }}}
    
```

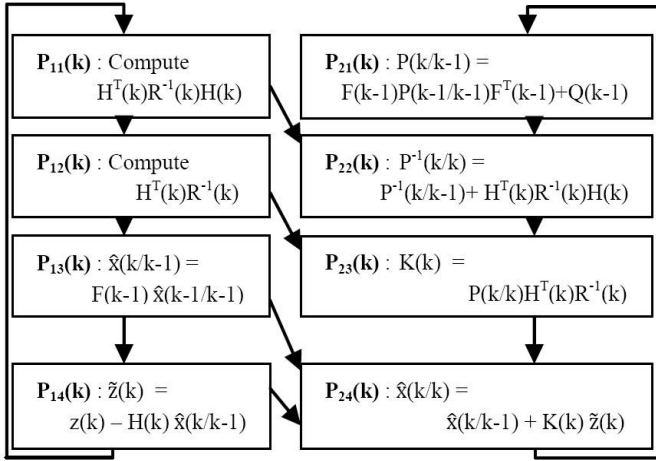


Fig. 4. Parallel pipeline structure for the state estimate.

Channels *COM1* and *COM2* corresponding to canal1 and canal2 in the Occam code, and channel *COM3* is a communication support from *T₁* to *T₂* through the physical connection *Link.2.in/out* (Fig. 5).

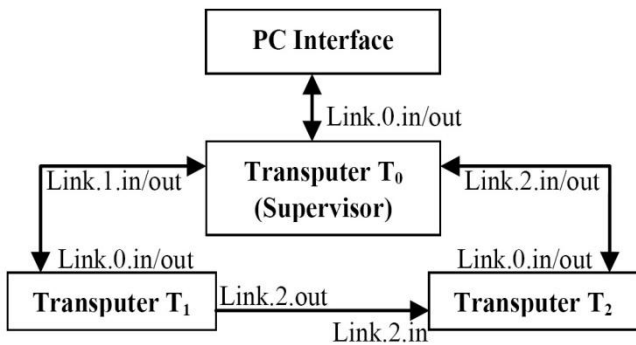


Fig. 5. SIMD transputer implementation.

In this work and in order to further improve the speed of updating the state estimate and taking full advantage of the two parallel pipeline structures of Fig. 3, a new combined SIMD/MISD (Single Instructions Multiple Data stream/

Multiple Instructions Single Data stream) configuration scheme is implemented (Fig. 6).

Clearly, Fig. 4 is a one time-iteration of a Kalman filtering process. The new implementation results, as will be seen in the next section, in a significant saving in processing time complexity over the previous one. Moreover, the larger the number of iterations is, the more efficient the proposed implementation is. For this new configuration, eight transputers are used to implement the eight segments of the two parallel pipelines of Fig. 4, all of them connected to a supervising transputer *T₀*.

The task of this transputer is the linking between the PC and the eight transputers but is mainly the supervising of different timing and input/output controls. If one looks for example at the two first segments of the right pipeline of Fig. 4, we can see if transputer *T₂₁* has finished executing process *P(k/k-1)* of Segment 1 at time instance *t₀*, its resulting output is passed to transputer *T₂₂* to execute process *P⁻¹(k/k)* of Segment 2 at time instance *t₁*, meanwhile transputer *T₂₁* is idle since process *P(k+1/k)* depends on process *P⁻¹(k/k)* being actually processed. Many such cases are present on this configuration and all of them will be seen in details in the next section.

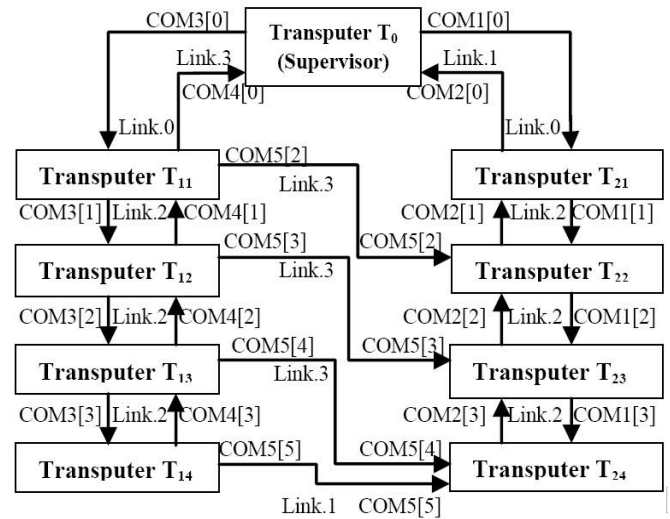


Fig. 6. Combined SIMD/MISD transputer implementation.

The Occam configuration for this proposed SIMD/MISD transputer implementation is the following:

```

{{{
--Define channel addresses:
  CHAN COM1[5], COM2[5], COM3[5], COM4[5],
  COM5[5]:
--sc. DispSup.occ
--sc. Main1.occ
--sc. Main2.occ
  PLACED PAR
    PROCESSOR 0 T8 --Transputer T800 (T0)
      PLACE COM1[0] AT Link.1.out:
      PLACE COM2[0] AT Link.1.in:
      PLACE COM3[0] AT Link.3.out:
      PLACE COM4[0] AT Link.3.in:
      DispSup(COM1[0], COM2[0], COM3[0],
        COM4[0], 0)
    PLACED PAR i=[1 FOR 4]
      PROCESSOR i T8 -- TranspT800(T2i)
    
```

```

PLACE COM1[i-1] AT Link.0.in:
PLACE COM2[i-1] AT Link.0.out:
PLACE COM1[i] AT Link.2.out:
PLACE COM2[i] AT Link.2.in:
PLACE COM5[i] AT Link.3.in:
PLACE COM5[5] AT Link.1.in:
Main1(COM1[i-1],COM2[i-1],COM1[i],
      COM2[i], COM5[i],COM5[5], i)
PLACED PAR j=[1 FOR 4]
PROCESSOR j T8 -- TranspT800(Tij)
PLACE COM3[j-1] AT Link.0.in:
PLACE COM4[j-1] AT Link.0.out:
PLACE COM3[j] AT Link.2.out:
PLACE COM4[j] AT Link.2.in:
PLACE COM5[j+1] AT Link.3.out:
Main2(COM3[i-1],COM4[i-1],
      COM3[j],COM4[j],COM5[j+1],j)
}}}
    
```

In this code, eight transputers treat the identical procedures *Main1* and *Main2*, each one consisting of both a systolic array process, in addition to a communication process to manage the control and data flow between neighboring transputers. Transputer T_0 (in Occam compilation named processor 0) is an interface and supervising transputer, displaying results received from the eight transputers and supervising also the communication between different transputers. These two processes are done using the *DispSup* procedure; a modified version of the *Display* procedure of [10].

The modifications consist of introducing a sequence for displaying results from the eight transputers instead of the two of the *Display* procedure and another sequence to control and supervise communications between transputers during each filtering pass at appropriate computing times.

V. PROCESSING TIME COMPLEXITY

The Occam timing instruction *TIMER* is used to evaluate time complexity for different implementations. The processing time duration T_r needed to perform one row, assuming the amount of time T_b and T_i required respectively for boundary and internal cells data processing, is given as:

$$T_r = T_b + (2n-1) T_i$$

And the time duration T_a of all rows of data (one pass of Table I) to actually pass through the array is given as:

$$T_a = (5n-1) T_r = (5n-1) [T_b + (2n-1) T_i]$$

In fact, the compound matrix of data is in general a $\{(4n-1) \times 2n\}$ matrix for n state variables. Thus, it takes $(4nT_r)$ to feed and compute the $(4n-1)$ rows and it takes $\{(n-1)T_r\}$ for the last row to propagate through the array and a result to appear and therefore a total of $\{(5n-1)T_r\}$ to compute one whole pass. In order to evaluate the time complexity for computing the eight passes of Table.1 (one filtering process), one can do it assuming initially a trivial implementation on one transputer. In this case, it is obvious that its total time complexity is $T_t = 8T_a = (40n-8)T_r$ and for m jobs (m time-iterations) $T_{tm} = 8mT_a$. However, when using the configuration of Fig.5 in conjunction with diagram of Fig. 4, the resulting time complexity is $T_t = 5T_a$ for one job and $T_{tm} = (4m+1)T_a \cong 4mT_a$ for m large. The transputer task affectation (T_1, T_2) and the space-time diagram for this implementation are shown in Table II and Fig. 7 respectively [10].

Moreover, and to do even better by taking advantage of the pipeline structures of Fig. 4, the combined SIMD/MISD transputer implementation of Fig. 6 has resulted in a significant saving in processing time complexity.

Thus, for m jobs, the time complexity is now $T_{tm} = (3m+2)T_a \cong 3mT_a$ for m large, resulting in a saving of mT_a representing 25% of the time complexity of the previous configuration (Fig. 4).

It is also clear that the larger the number of processed jobs is, the more computational speedup and processor usage efficiency are obtained.

Table III and Fig. 8 show respectively the transputer task affectation and the space-time complexity diagram for this new implementation (SIMD/MISD).

In Table II and Table III, all t_i 's have the same duration T_a (same process) and empty boxes are to show that respective transputers are idle. Fig. 7 and Fig. 8 show the space-time diagrams for m jobs and empty boxes are also to show that respective transputers are idle. The use of the Occam *TIMER* instruction has resulted in the same processing time for both the boundary and internal cells $T_c = T_b = T_i$. Thus, $T_r = (2n)T_c$ and $T_a = 2n(5n-1)T_c$.

In our hardware implementation using T800 transputers configured and compiled with Occam language protocol proposed in Fig. 8, we have obtained a processing time $T_c = 92msec$. Also, the use of more performing transputers will certainly lower the time complexity and therefore speedup the processing [11].

TABLE II: TRANSPUTER TASK AFFECTATION FOR THE SIMD TRANSPUTER IMPLEMENTATION

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	...
T2		$P_{21}(k)$	$P_{22}(k)$	$P_{23}(k)$	$P_{24}(k)$	$P_{21}(k+1)$	$P_{22}(k+1)$	$P_{23}(k+1)$	$P_{24}(k+1)$	$P_{21}(k+2)$	$P_{22}(k+2)$	$P_{23}(k+2)$
T1	$P_{11}(k)$	$P_{12}(k)$	$P_{13}(k)$	$P_{14}(k)$	$P_{11}(k+1)$	$P_{12}(k+1)$	$P_{13}(k+1)$	$P_{14}(k+1)$	$P_{11}(k+2)$	$P_{12}(k+2)$	$P_{13}(k+2)$	$P_{14}(k+2)$

VI. CONCLUSION

In this work, a new combined SIMD/MISD configuration for parallel Kalman Filtering is proposed. The configuration

of eight parallel transputers for Kalman filter operations shows more efficiency in speed up of the processing time (a saving of mT_a representing 25% of the time complexity of the previous configuration) and in speed-up of the updating of the state estimate of the parallel Kalman filtering. The use

of Occam language to implement this configuration has been proved to be suitable and really efficient for partitioning relative processes and data. High performances

have been attained in terms of computational speedup and bandwidth. Moreover, the larger the number of processed jobs is, the higher the performances are.

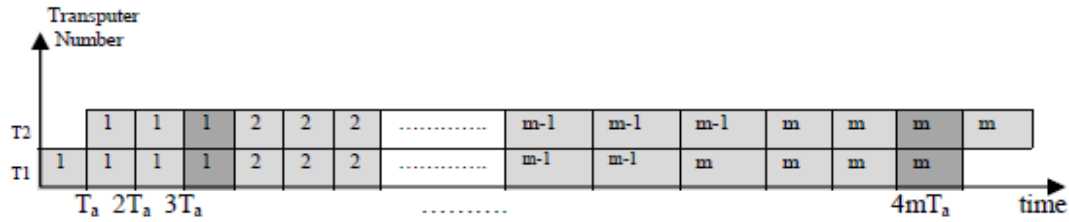


Fig. 7. Space-time diagram for the SIMD implementation.

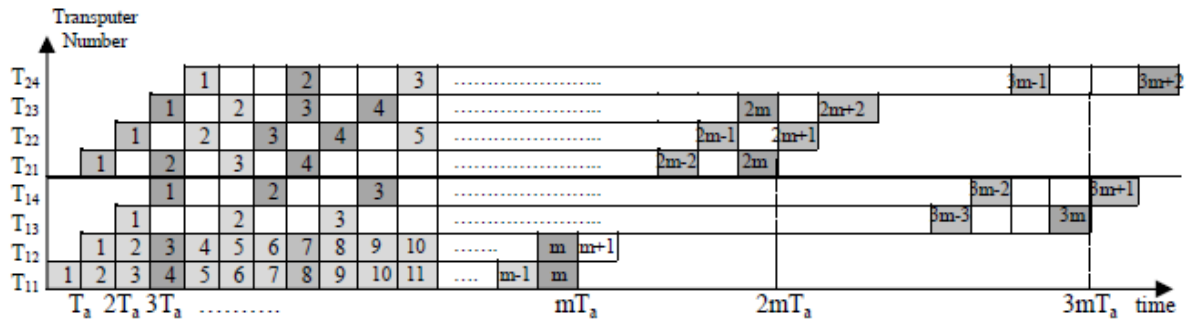


Fig. 8. Space-time diagram for the SIMD/MISD implementation.

TABLE III: TRANSPUTER TASK AFFECTATION FOR THE COMBINED SIMD/MISD TRANSPUTER IMPLEMENTATION

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	...
T_{24}					$P_{24}(k)$			$P_{24}(k+1)$			$P_{24}(k+2)$...
T_{23}				$P_{23}(k)$		$P_{23}(k+1)$		$P_{23}(k+2)$		$P_{23}(k+3)$		$P_{23}(k+4)$...
T_{22}			$P_{22}(k)$		$P_{22}(k+1)$		$P_{22}(k+2)$		$P_{22}(k+3)$		$P_{22}(k+4)$...
T_{21}		$P_{21}(k)$		$P_{21}(k+1)$		$P_{21}(k+2)$		$P_{21}(k+3)$		$P_{21}(k+4)$		$P_{21}(k+5)$...
T_{14}				$P_{14}(k)$			$P_{14}(k+1)$			$P_{14}(k+2)$...
T_{13}			$P_{13}(k)$			$P_{13}(k+1)$			$P_{13}(k+2)$			$P_{13}(k+3)$...
T_{12}		$P_{12}(k)$	$P_{12}(k+1)$	$P_{12}(k+2)$	$P_{12}(k+3)$	$P_{12}(k+4)$	$P_{12}(k+5)$	$P_{12}(k+6)$	$P_{12}(k+7)$	$P_{12}(k+8)$	$P_{12}(k+9)$	$P_{12}(k+10)$...
T_{11}	$P_{11}(k)$	$P_{11}(k+1)$	$P_{11}(k+2)$	$P_{11}(k+3)$	$P_{11}(k+4)$	$P_{11}(k+5)$	$P_{11}(k+6)$	$P_{11}(k+7)$	$P_{11}(k+8)$	$P_{11}(k+9)$	$P_{11}(k+10)$	$P_{11}(k+11)$...

REFERENCES

- [1] J. Hinton and A. Pinder, *Transputer Hardware and System Design*, Prentice Hall, 1993
- [2] H. R. Arabnia and M. A. Oliver, "A transputer network for fast operations on digitized image," *Computer Graphics Forum*, Oct. 2007, vol. 8, issue 1, pp. 3-11.
- [3] M. Tanaka, N. Fukuchi, Y. Ooki, and C. Fukunga, "A design of transputer core and its implementation in an FPGA," *Communication Process Architecture*, IOS Press, 2004
- [4] J. E. L. Hollis and T. E. Crunk, "Transputer implementation of interpolators for radar image transformation," *Microprocessors & Microsystems*, 1995, vol. 9, issue 4, pp. 179-183.
- [5] H. G. Yeh, "Systolic implementation on kalman filters," *IEEE Trans. & Acoustic, Speech & Signal Proc.*, Sept. 1988
- [6] J. G. Nash and S. Hansen, "Modified fadeev algorithm for matrix manipulation," in *Proc. SPIE Conf., San Diego, CA*, 1984, pp. 39-46.
- [7] W. M. Gentelman and H. T. Kung, "Matrix triangularization by systolic arrays," *SPIE*, vol. 289, 1981

- [8] G. Jones, "Programming in Occam," *Programming Research Group, Technical Monograph PRG-43*, 1985
- [9] Occam Implementation Manual, INMOS Limited, 1988
- [10] N. Taleb, A. Seddiki, R. Menezla, and M. F. Belbachir, "Transputer-based system for parallel kalman filtering," *International Journal of Applied Mathematics*, vol. 5, pp. 565-584, 2000
- [11] Q. Li and N. Rishe, "A transputer T9000 family based architecture for parallel database machines," *ACM SIGARCH Computer Architecture News-Special issue on input/output parallel computer systems*, vol. 21, issue 5, pp. 55-62, Dec. 1993.



Ali Seddiki received an Elect.-Eng. degree, Master degree, from Djillali-Liabes University, Sidi-Bel-Abbes, Algeria, respectively in 1994, 1997, and a PhD degree in Electrical Engineering in 2006. He is currently an assistant professor at the Electronic Engineering Department of the University of Sidi-Bel-Abbes. He is a research scientist in Telecommunication and Digital Signal Processing Laboratory. His research interests include signal and medical image applications, wavelet applications, source coding, and image analysis.



Nasreddine Taleb received a M.Sc. degree in Computer Engineering from Boston University, an Elect.Eng. degree from Northeastern University, Boston, and a PhD degree in Electrical Engineering from Djillali Liabes University, Sidi Bel Abbas, Algeria. He is currently a Professor at the Electronic Engineering Department of the University of Djillali Liabes, where he is also a Senior Research Scientist

and the Director of the “Communication Networks, Architecture, and Multimedia” Laboratory. His principal research interests are in the fields of digital signal and image processing, image analysis, medical and satellite image applications, pattern recognition, and advanced architectures for the implementation of DSP/DIP applications.



Driss Guerchi is currently an associate professor and coordinator of the telecommunication program at the Canadian University of Dubai. He was an assistant Professor at the College of Information Technology, UAE University from 2001-2009.

He received his PhD in Telecommunications from the National Institute of Scientific Research, and his Master degree in Physics from Canada, in 2001, University of Quebec at Montreal. His research interests include speech coding, speech steganography and signal processing.