Dependence Vectors and Fast Search of Systolic Mapping for Computationally Intensive Image Processing Algorithms

Bala Tripura Sundari B[1]

Abstract- **2-D convolution in image processing and Full Search Block Motion (FSBM) estimation used in a H.264 video encoder, are highly data intensive and computationally intensive algorithms. Such algorithms require high memory access bandwidth due to repeated memory access. They are represented as nested do loop algorithms to enable systolic mapping. Mapping is used to facilitate the extraction of parallelism along with efficient data reuse. To enable the above, the dependence vector formulation and extraction of dependencies between iterations have been used. To implement the former the searching of scheduling vector t and Processor Matrix P is performed to form the mapping transformation matrix M. The focus of our work is the extraction of the dependence vectors from the application algorithm, followed by the search of the mapping matrix M, where a novel method of finding t vector has been used. This saves the search time as compared to the widely used exhaustive search methods. The resultant M matrix is used to arrive at the various design trade – offs. The method is applied to 2-D filtering algorithm and (FSBM) which act as good test cases for nested loop algorithms. The architecture is simulated and synthesized using Mentor Graphics tools and targeted to Virtex FPGA.**

Index Terms— **Systolic Mapping, Dependence Vectors, Data Reuse, Iteration Space, Mapping Matrix, Nested loop algorithm, Systolic Array, FSBM, 2-D Filtering.**

I. INTRODUCTION

Deeply nested loop algorithms are computationally intensive algorithms and exhibit repeated set of patterns of operations, with high data reuse. They are suitably represented in Uniform Recurrence Equation (URE) form. They become good candidates for massive parallel implementation. [1]-[4] & [6]. Maximum use of input ports ensuring maximum data reuse and optimal designs to maximize throughput are the criteria aimed at in evolving the designs.

A. Mapping

Any function which allocates a processor and a time slot for each iteration can be used as mapping function. The proper choice of the mapping matrix M (1) is an important step. Here M is the projecting matrix that maps the iteration space to a processor at a particular time [5]. The dependency between the two iterations I_1 and I_2 can be represented by the difference between them. I_1 and I_2 are vectors of equal dimensions which is equal to the order *n* of the n-D nested loop algorithm.

$$M=\begin{bmatrix} P \\ t \end{bmatrix} \qquad (1)$$

The various constraints such as design, causality, and mapping constraints are discussed in the literature. Heuristic search methodology is used to arrive at an optimum design [5]. Development of architectures from parallel algorithms using cut-set systolisation process is analyzed [10]. The derivation of feasible mapping is done by identifying formal criteria to be satisfied by both the original sequential algorithm and the proposed transformation function [11]. The problem of optimally mapping uniform DAGs to systolic arrays using retiming and an affine timing function is developed [7], [8] & [9]. Parametric VLSI architectures and Systolic architectures for FSBM have been implemented [12], [13].

The FSBM algorithm is a six level nested do loop algorithm. The salient features of the work presented here are a) Formulation of a 4-Dimensional (4-D) algorithm for FSBM as shown in listing1. b) The 4-D algorithm is used to illustrate the formation of the dependence vectors in a reduced index space - 4-D index in section II. c) The extraction of dependence vectors and d) reduction of search space for the scheduling vector t by forming a set of dependency constraints e) the above features are implemented for a 4-D nested loop edge detection algorithm and 6-D nested loop FSBM algorithms. The organization of the paper is as follows: Section III presents the search method adopted for the edge detection algorithm, and the mapping results and section IV presents the methodology and mapping results for FSBM algorithm.

II. DEPENDENCE VECTORS

A. Formulation of Dependence vectors for Full Search Block Motion (FSBM) Algorithm

We consider the conventional six level nested do loop algorithm for FSBM estimation [1]. The formulation of the dependence vector set is brought out through the reduced 4-D FSBM algorithm which forms a reduced 4-D index space.

Manuscript received May 17, 2010; revised October 15, 2010.
[1]Assistant Professor, Departmant of ECE, Amrita School of Engineering, Amrita Vishwa Vidyapeetham, Coimbatore -641 105, India, balasrikanth2003@yahoo.com

B. Dependence vectors for a reduced index space - 4-D FSBM

Consider the current frame pixels –(*c* frame) of size 8X8 pixels in figure 1. This frame is divided into sub frames or blocks of size NXN. (N = 4). The blocks are scanned row wise. The sequence of blocks is numbered as follows: h is the maximum number of blocks in a row and v is the maximum number of blocks in a column. The sequence number for the blocks is indicated by a new variable h_v, reducing the two variables h and v to h_v. If the current frame pixel is 36 presently, then its corresponding search frame pixels are shown by the enclosed dotted rectangle of pixels for p =N/2 = 2. Hence the size of the search frame becomes $(2*p+1)^2$ =25 pixels. The search frame data are represented as the *s* pixels . The search variables m and n indicate the all possible search directions as in the FSBM estimation algorithm –listing 2 . They act as the search direction variables in the 6-D index space. The search variables m, n are the additional variables which indicate the search pixel directions for the current frame pixel c_{33} as shown in the figure 2. Here m, n vary from 1 to 5 when p= 2. These search variables m and n are reduced to a single variable called the comparison number 1 to 25 as shown in the fig. 2.

C. Dependence Graph for formulation of Dependence vectors (D_{vy})

When p = 2, the number of search frame data required for comparison for one current frame pixel is $(2*p+1)^2$ = 25. The dependence vector is formulated manually from the Dependence Graph (DG) representation in figure 2. The numbers on the nodes represent the comparison numbers of the search frame that are to be compared for a single current frame pixel. The pixels within a block are scanned column wise. So the next current frame pixel is c_{34}.

```
For h_v = 1: N_h*N_v,
  MV (h_v) = 0;
  Dmin(h_v) = ∞;
  For p = 1: (2p+1)^2, mad (p) = 0;
      For i = 1: N,      For j = 1: N,
            mad (p) = mad (P) + |x (h_vN^2 + i, h_vN^2 + j ) – y (h_vN^2 + i+p, h_vN^2 + j+p)|;
      End j,   End i
If (D_min(h_v) > mad (p))
      D_min(h_v) = mad(p)
      MV (h_v) = p;

endif
   End p,End h_v
```

Listing 1 4-level FSBM architecture

From the fig. 2 and fig. 3 the dependence vector for *s* data propagation is calculated as follows: when data flows from the 13th to the 8th comparison. D_v for s-frame data is termed as D_{vs1}.

$D_{vs1} = h_v, p_{new}$, i, j = (0,13,3,3) – (0, 8,3,4).
= 0, 2*p+1,0,1.

D_{vs2} = 1, 2*p+1, N-1, 0 between adjacent blocks.

D. Dependence Graph (DG) for c data propagation (D_{vc})

$D_{vc1} = h_v ,p_{new}$,i,j: 0,1,3,3 to 0,2,3,3 = 0,1,0,0;

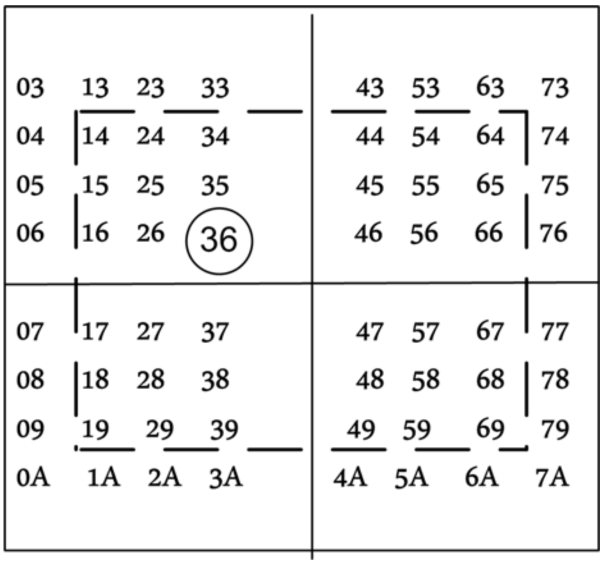

Fig. 1 c- frame data c36 (shown encircled), and the corresponding search frame pixels –shown within the dotted frame.

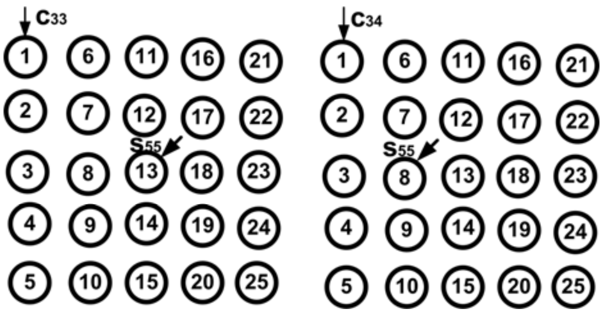

Fig. 2 Dependence graph for *c* frame pixel 33 & 34, search Frame pixel s_{55} –shows search frame data propagation and data reuse.

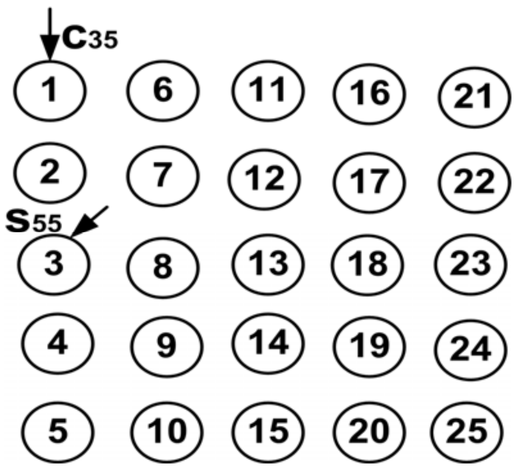

Fig. 3 DG to determine *s* frame data dependence for C_{35}

From the above Dependence graph, we know that c is to be propagated to all nodes of the DG and hence the D_v for c is given as shown in the D_v equation for FSBM and Dvc2 is given as follows:

D_{vc2} = 0,0,3,3 to 0,5,3,3 = 0,(2*p+1),0,0.

E. Dependence Vector for MAD data accumulation & propagation (D_{vMAD})

TABLE I D_{VMAD} FOR 1 MACROBLOCK

MAD elements	h_v	p_{new}	ij ●
03-25,04-26, 05-27,06-28	0	0	03,04,05,06
13-35,14-36, 15-37,16-38	0	0	13,14,15,16
23-45,24-46, 25-47,26-48	0	0	23,24,25,26
33-55,34-56, 35-57,36-58	0	0	33,34,35,36

The Dependence Vector for $D_{v\,(MAD)}$ is given as

$D_{v\,(MAD)}$ = [0 0 0 1; 0 0 1 –(N-1)]

F. Dependence Graph for D_{vDmin} data propagation (D_{vDmin})
The Dependence vector for D_{min} shown above arises out of the fact that D_{min} has to be propagated to the next sub-frame and is gives as $D_{v(Dmin)}$ = [1 0 0 0]. All the above dependence vectors are combined to form the dependence vector matrix as shown below. The last row gives the variables. The above method explains the formulation of the dependence vector set which is an important step in the mapping process

$$\text{DVFSBM} = \begin{array}{cccccccc} 0 & 0 & 1 & 0 & 0 & 0 & h & 1 \\ 1 & 2p+1 & 2p+1 & 0 & 0 & 2p+1 & 1 & 0 \\ 0 & 0 & N-1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & -(N-1) & 0 & N-1 & 0 \\ x1 & y_1 & y_2 & MAD_{1,} & MAD_2 & x2 & y3 & D_{\min} \end{array}$$

III. FINDING DEPENDENCIES IN A NESTED LOOP ALGORITHM USING AN AUTOMATED METHOD

The nested loop algorithm is read to form the index space. The variables used in algorithm are added to the variables list. We consider (2) as the main loop expression Written in post operator form as in (3).

$$c(i\,,j)\;\; c(i\,,j)\;\; a(i\,,k)\;\; b(k\,,j)\; *\; +\; = \qquad (2)$$

The procedure adopted is given below:

- Select the first element from the iteration space which represents the first iteration. Substitute the variables in the body of the loop by the iteration value considered. For the iteration (1, 1, 1).

$$c(1\,,1)\;\; c(1\,,1)\;\; a(1\,,1)\;\; b(1\,,1)\; *\; +\; = \qquad (3)$$

- Add the operand which is at the extreme left in the post-operand form of the loop body to the set of variables being updated along with the remaining variables to the set of variables being used. Now U={c (1, 1)} and C= {c (1, 1), a (1, 1), b (1, 1)} for the iteration (1, 1, 1).
- Check for any common variable that is being used in the current iteration (I_2)and the previous iteration (I_1).If any common variable is found, it means that this current iteration depends on the previous iteration. It also determines the dependencies for the computational variables similar to the previous case.
- Add this vector to the dependency vector set D.
  Select the next iteration and continue till the end. The methodology and the results of the dependencies, search results of t and P matrix, and the design trade-off of the hardware is discussed in the following sections.

A. Mapping of the edge-detection algorithm

The 2-D convolution and many morphological operations require data reuse and regular iterative computation. We assume that this 4-level algorithm is targeted to an architecture that is two dimensional (Dim=2). The size of the t vector is 1× 4 and that of P matrix is 2 ×4. Hence the M matrix has a dimension of (Dim+1, n) = (3, 4). There are 12 elements in each possible M matrix. If each value in M has 6 possible values during searching then a total of 6^{12} M matrices have to be checked for the constraints and the objectives.

The search space is significantly reduced as compared to heuristic search methods already used, by first directly finding the Scheduling vector t that satisfies the causality constraints arising out of the dependencies. Then search for processor-Allocation matrix P which satisfies the mapping constraint. The listing 2 shows the 4 level edge detection algorithm. The algorithm finds the edges in an image of size (m × n) = (5 × 5) and the mask size of 3×3. The index space for this algorithm is (i, j, k, l) and the iteration space I is given by:

I= {(I, j, k, l): 0<= I <15, 0<= j <15, 0<= j< 2, 0<= l <2}.

The following is the listing of the edge-detection algorithm.

```
For(i=0;i<m;i++)
For(j=0;j<n;j++)
{
O[i,j]=0;
For(k=0;k<3;k++)
For(l=0;l<3;l++)
O[i,j]=O[I,j]+I[i+k,k+l]*w[k,l];
}
```

Listing 2 Edge-Detection Algorithm

The dependencies found in this algorithm due to both computational and propagation constraints are listed below.

TABLE II DEPENDENCE VECTORS EXTRACTED AUTOMATICALLY BY THE SEARCH ALGORITHM FOR THE EDGE-DETECTION ALGORITHM

Dependence (D)	Variable	Type of the dependence & Propagation delay assumed
(0,0,0,1)	O	Computational(3)
(0,0,1,-2)	O	Computational(3)
(0,1,0,0)	W	Propagation(1)
(1,-4,0,0)	W	Propagation(1)
(0,1,0,-1)	I(0,1)	Propagation(1)
(1,0,-1,0)	I(1,0)	Propagation(1)
(1,-1,-1,1)	I(1,1)	Propagation(1)
(1,-2,-1,2)	I(1,2)	Propagation(1)

In table II the delays shown in the brackets are assumed depending on data dependencies and computational delays.

B. Finding t vector

The t-vector should satisfy the causality constraint which is given by the expression $t \times D_v^T > [dd]$. Here D is the dependence vectors set and dd is a vector which represents the delays (given in table II) associated with computational variables and propagation variables. This helps us to use the actual computational and propagation delays associated with the hardware. The size of t matrix is (1×n). Let t = (a b c d). Now substituting each dependence vector in the causality constraint we get the following inequalities.

TABLE III INEQUALITIES FOR FINDING T MATRIX

d>3	b-d>1
c-2d>3	a-c>1
b>1	a-b-c + d>1
a-4b>1	a-2b-c+2d>1

Solving these inequalities for a general solution, using MATLAB we obtain the scheduling matrix t directly. The first solution for the above inequalities is given by:

t= (a b c d) = (8 2 7 2).

An m file is used to solve the inequalities in the table III using pseudo –inverse method. So here we are directly finding the solution for t matrix without searching for it which reduces the search from 6^{12} matrices to 6^8 matrices so we can skip 6^4 matrices. This reduces the search time as compared to the heuristic search methods.

C. Searching for P-Matrix

After finding t matrix we have to search for the P matrix which satisfies the mapping constraint as explained in [11]. For the edge detection algorithm, the P matrix is a (2×4) matrix. One of the valid P matrices is given by:

$$P=\begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}$$

Thus one of the valid M matrix is given by

$$M=\begin{bmatrix} 8 & 2 & 7 & 2 \\ 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}$$

This reduces the search time, when compared to both t and P matrices being searched together. The search time comparisons are summarized in the table 12 in terms of complexity.

D. Mapping of Index space and Dependence vectors

The mapping results for the above M matrix are given below. The First iteration space maps to the processor-time space and the dependence vectors map to Edge-delay space as shown in table IV.

TABLE IV RESULTS OF MAPPING OF THE DEPENDENCE VECTOR SET

Variable	Edge	Delay
(0,0,0,1) <O>	(0,0)	2
(0,0,1,-2) <O>	(0,0)	3
(0,1,0,0) <W>	(-1,-1)	2
(1,-4,0,0) <W>	(4,4)	0
(0,1,0,-1) <I>	(-1,-1)	0
(1,0,-1,0) <I>	(0,0)	1
(1,-1,-1,1) <I>	(1,1)	1
(1,-2,-1,2) <I>	(2,2)	1

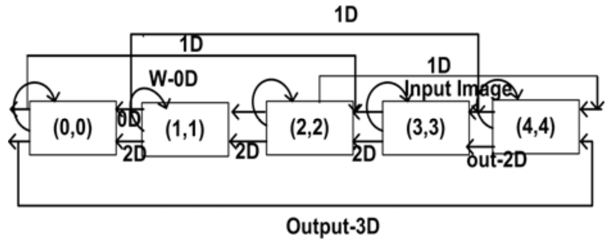

Fig. 4 Architecture obtained using the mapping for the edge detection algorithm

E. Mapping Results

The cost function is defined as (5) which can be used as a filter for selecting architecture according to the target hardware mapping technology limitations.

Cost = a×processors + b×cycles + c××I/O ports (4)

Here a, b, c are the scalar coefficients which represents the weights for the corresponding costs to minimize the overall cost function.

F. Mapping for different image and window sizes

TABLE V MAPPING RESULTS FOR EDGE DETECTION

	a=1,b=9, c=1	a=9,b=1, c=1	a=1,b=1, c=9
Processors	9	5	7
Cycles	53	59	55
IO ports	7	5	5

TABLE VIA CASE 1. MAPPING RESULTS: INCREASING IMAGE SIZE WITH NO CONSTRAINT OF CONSTANT PROCESSORS

Image size	Window size	Processors	Cycles
5×5	3	5	53
6×6	3	9	55
8×8	3	14	56

TABLE VIC CASE 3. MAPPING RESULTS INCREASING IMAGE SIZE WITH CONSTANT CYCLES

Image size	Window size	Processors	Cycles
5×5	3	5	53
6×6	3	9	55
8×8	3	14	56

TABLE VID CASE 4. CHANGE IN MAPPING RESULTS WITH INCREASING WINDOW SIZE AND CONSTANT IMAGE SIZE WITH CONSTRAINT OF CONSTANT PROCESSORS

Image size	Window size	Processors	Cycles
5×5	3	5	53
6×6	3	5	61
8×8	3	6	72

TABLE VIE CASE4. CHANGING IN MAPPING RESULTS WITH INCREASING IMAGE SIZE WITH THE CONSTRAINT OF CONSTANT PROCESSORS

Image size	Window size	Processors	Cycles
5×5	3	5	53
6×6	3	9	55
8×8	3	14	56

We consider the edge-detection algorithm for higher image sizes, the number of iterations increases which leads to increase in the number of processors or increase in the number of cycles.

G. Performance

If the input image considered is of 5×5 and the window size is 3×3 then a total number of 225 iterations are required when done sequentially with one processor to implement the algorithm. The systolic array design performs the required computation on the given image size in 53 cycles (Table V). This is verified by the architecture simulated in Modelsim 5.8 (Mentor Graphics tool) and the memory read operations are reduced from 225 to 101 due to Propagation of the input image data between the processor.

IV. MAPPING OF FULL SEARCH MOTION ESTIMATION ALGORITHM

The Full Search Motion Estimation algorithm considered is a 6- level algorithm [4] for finding the motion vectors between two successive frames in a video is shown in listing [2].

```
For h=0 to ($N_v$-1)
For v=0 to ($N_h$-1)
{{
$D_{mn}$ (h,v)=∞
For m=0 to (2*p)
{For n=0 to (2*p)
{MAD (m,n)=0
for i=0 to (N-1)
{
for j=0 to (N-1)
{MAD(m,n)=MAD(m,n)+x(h*N+i,v*N+j)-y(h*N+i+m-p,v*N+j+n-p)
}}
If ($D_{mn}$(h,v)>MAD(m,n))
$D_{mn}$(h,v)=MAD(m,n)
MV (h,v)=(m-p,n-p)
}}}}
```

Listing [3] 6-level FSBM Algorithm

If we let (a b c d e f) to be the required t Matrix, then the system of constraints obtained from the causality constraints for the dependence vectors are listed in the table VII. The propagation and computation time for D_{min} is considered as

5 and that for MAD data to be 4. The propagation time for c data and s data are assumed to be 1 as in the table VII.

TABLE VII INEQUALITIES FOR FINDING S MATRIX

d >5	d-f>1
c-4d>5	c-2d-e+2f>1
f>4	d+2e-f>1
e-3f>4	d-f>1
c-e>4	c-2d+e-2f>1
b-3e-3f>4	c-d-e+f>1
d>1	c+d-e+f>1
d-4e>1	c-4d-e+4f>1

TABLE VIII DEPENDENCE VECTORS FOR FSMB ALGORITHM

Variable	Dependence	Type (C or P)	D-elements $=D_v \times t$
D_{min}	0 0 0 1 0 0	C	0
D_{min}	0 0 1 -4 0 0	C	5
MAD	0 0 0 0 0 1	C	0
MAD	0 0 0 0 1-3	C	0
MAD	0 1 0 0 -3-3	C	0
MAD	1-1 0 0 -3 -3	C	4
X(0,0)	0 0 0 1 0 0	P	0
X(0,0)	0 0 0 1 -4 0	P	0
X(1,2)	0 0 1 0 1 0	P	2
X(2,1)	01 0 0 -3 -3	P	0
Y(0,0)	0 0 0 1 0 -1	P	0
Y(0,0)	00 1 -2 -1 2	P	2
Y(0,1)	0 0 0 1 2 -1	P	0
Y(0,1)	0 0 0 1 0 -1	P	0
Y(0,1)	00 1 -3 -1 3	P	2
Y(1,0)	0 0 0 1 0 -1	P	0
Y(1,0)	00 1 -2 -1 2	P	2
Y(2,3)	0 0 0 1 0 -1	P	0
Y(2,3)	00 1 -2 1 -2	P	2
Y(2,3)	0 0 0 1 0 -1	P	0
Y(2,4)	00 1 -1 -1 1	P	2
Y(1,0)	0 0 0 1 0 -1	P	0

One of the solutions for these in-equalities is given by t = (a b c d e f) = (2, 0, 2, 0, 0, and 0) and the P matrix can be searched to get the M matrix which maps onto 16 processors and 196 cycles, which is given below.

$$M=\begin{bmatrix} 2 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

A Architecture

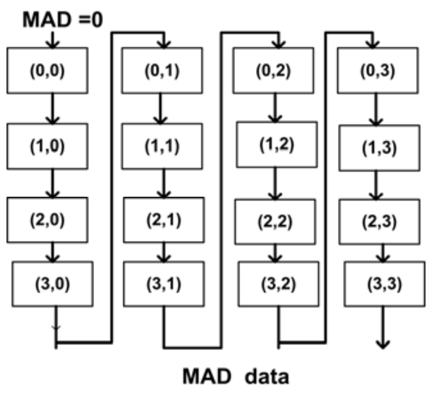

Fig. 6 Propagation of MAD data

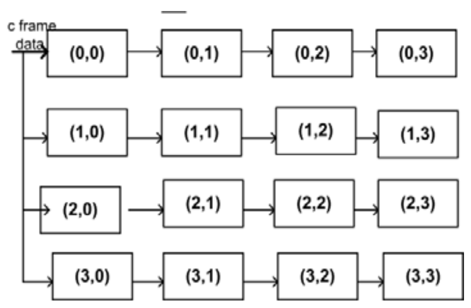

Fig. 7 Propagation of c frame data

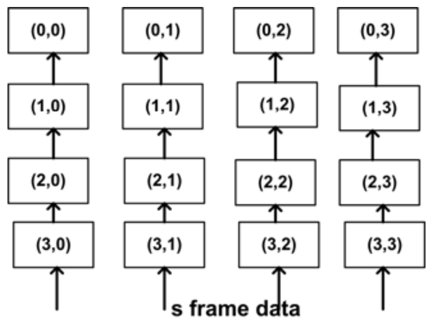

Fig. 8 Propagation of s frame data with 2 Delays (not labeled in the diagram)

The mapping results for different cost functions are given in table IX as per expression (4).

TABLE IX MAPPING RESULTS FOR FSBM ALGORITHM

	a=1,b=9,c=1	a=9,b=1,c=1	a=1,b=1,c=9
Processors	25	16	16
Cycles	124	196	177
IO ports	7	9	4

B. Hardware Implementation

The architecture is implemented in FPGA- virtex-5 using verilog HDL. The Device utilization for the edge detection algorithm and the motion estimation algorithm is given in table X. The synthesis results of synthesis tool Xilinx ISE are given in the table XB and XC.

TABLE XA COMPARISON BETWEEN THE DEVICE UTILIZATIONS FOR EDGE-DETECTION(5X5 IMAGE, 3X3 WINDOW SIZE AND FSMB ALGORITHMS: N=4, P=2

Algorithm	LUT * utilization	FF util ization	IOB
Edge-detection	131	71	55(24%)
FSBM	172	88	48(21%)

TABLE XB DEVICE UTILIZATION OF THE CONTROL UNIT FOR THE MAPPED ALGORITHMS

Algorithm	LUT utilization	FF utilization	IOB
Edge-detection	96	60	94(42%)
FSBM	132	54	66(29%)

TABLE XC DEVICE UTILIZATION OF THE PROCESSOR ARRAY FOR THE MAPPED ALGORITHMS

Algorithm	LUT utilization	FF utilization	IOB
Edge-detection	153	98	41(18%)
FSBM	205	110	62(28%)

Total Device utilization of the mapped Nested loop algorithms targeted to FPGA virtex5 family.

* LUT=Look Up Table, **FF=Flip Flop, ***IOB=Input Output Blocks.

C. Complexity

The complexity is calculated and presented in table XI, which shows the speed up of the search process to arrive at the Mapping matrix M.

TABLE XI COMPLEXITY CALCULATIONS FOR VARYING VALUES OF N

n	Np_{n-1} number of possible values of elements of t, P	Other values	Total possible #	Number of Possible##	Our Method
2	$2p_1=2$; $2p_2=1$	0,1,-1	6	2×3 ;6^6	0,1,-1,u_i,u_j = 5, possible values; 5^6; ** =54
3	$3p_2=3$;3p3 =1	0,1,-1	7	$n\times3$;3×3 elements $=7^9$	0,1,-1,u_i,u_j, u_k=6 possible values; 6^9;6^6
4	$4p_3=4$,4p2=6,4 $p_1=4$	0,-1,1	17	$n\times3=12$; 17^{12}	7^{12} values; ; ** 7^8
6	$6p_5=6$,$6p_2=15$,6 $p_3=15$,$6p_2=90$	0,-1,1	App= 100	$n\times3=18$; 100^{18}	9^{18} ; ** 9^{12}

# values; ##matrices

**=3 rows in M matrix

** further reduced to

n - n level nested loop algorithm

V. CONCLUSION AND FUTURE WORK

The formulation and extraction of dependence vectors is an important step leading to efficient parallelization by mapping. The extraction of dependencies, for both propagation and computational variables between different intermediate iterations of the algorithm are extracted from the algorithm. This has been tested on two benchmarks on a suitable iteration space. The scheduling vector t is searched by using the dependency constraint and the causality constraint. The execution time for search using our method is far below the heuristic search methods adopted. This has reduced the overall search time to arrive at suitable mapping matrix M. The mapping results are verified for edge-detection and FSBM algorithms by simulating the architecture using Mentor Graphics Tool. Modelsim 5.8 and the synthesis results for targeting the design on to a virtex- 5 FPGA is presented.

ACKNOWLEDGMENT

The Author thanks the management of Amrita School of Engineering, Amrita Vishwa Vidyapeetham, Coimbatore, Tamil Nadu, India, for the facilities provided to carry out this work in the VLSI Lab.

REFERENCES

[1] Dongming Peng and Lu M, "On Exploring Inter-Iteration Parallelism Within Rate-Balanced Multirate Multidimensional DSP Algorithms", IEEE Transaction On Very Large Scale Integration (VLSI) Systems, Vol. 13, No. 1,2005, pp. 106-125.

[2] Keshab Parhi K, "VLSI Signal Processing", John Wiley, 1991.

[3] Khalili A.J.AI, "Synthesis of Systolic Arrays from Single Assignment Algorithm", IEEE Transactions on Signal Processing, 1995.

[4] Kung S.Y, "VLSI array processors", IEEE ASSP Magazine, Vol.2, No.3, 1985, pp. 4-22.

[5] Komarek T. and Pirsch P, "Array architectures for block matching algorithms", IEEE Trans. Circuit Systems, vol. 36, 1989, pp. 1301–1308.

[6] Lee P. and Kedem Z. M. "Synthesizing linear array algorithms from nested for loop algorithms", IEEE Transactions in Computers, vol. 37, 1998, pp. 1578–1598.

[7] Nelson Luiz Passos and Edwin Hsing-Mean Sha, "Achieving Full Parallelism Using Multidimensional Retiming", IEEE Transactions on Parallel and Distributed Systems, Vol. 7, No. 11, 1996, pp. 1150-1163.

[8] Naresh Maheshwari and Sachin Sapatnekar, "Efficient Retiming of Large Circuits", IEEE Transaction On Very Large Scale Integration (VLSI) Systems, Vol. 6, No. 1, 1998, pp. 74-83.

[9] Patrice Quinton. A, Djamegni C.T, Sanjay Rajopadhye, Tayou Risset, Tchuente M, "A Reindexing based Approach towards mapping affine schedules onto parallel embedded systems", Journal of parallel and distributed computing, 2009, pp. 1-11.

[10] Prasanna Kumar V.K.and chen Tsai Yu, "On synthesizing optimal family of linear systolic arrays for matrix multiplication", IEEE Transactions on Computers, Vol. 40, No. 6, 1991, pp. 770-774.

[11] Surin Kittitornkun and Hen Hu Y, "Mapping Deep Nested Do-Loop DSP Algorithms to Large Scale FPGA Array Structures", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 11, No. 2, 2003, pp. 208-217.

[12] Vos L. D. and Stegherr M, "Parameterizable VLSI architectures for the full-search block-matching algorithm", IEEE Transactions Circuit Systems, vol.36, 1989, pp. 1309–1316.

[13] Yeo H.and Hu Y. H, "A novel modular systolic array architecture for full-search block matching motion estimation", IEEE Transactions on Circuit Systems Video Technology, vol. 5, 1995, pp. 407–416.